DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Distribution Category:
Mathematics and Computers (UC-32)

ANL--87-23

DE88 000137

ANL-87-23

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

The SUPRENUM Communications Subroutine Library for Grid-oriented Problems

*Rolf Hempel**

Mathematics and Computer Science Division

June 1987

This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38.

*Permanent Address: Gesellschaft für Mathematik und Datenverarbeitung mbH, Postfach 1240, 5205 St. Augustin, Federal Republic of Germany.

MASTER

LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

Contents

The SUPRENUM Communications Subroutine Library for Grid-oriented Problems

*Rolf Hempel**

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439-4844

Abstract

In the application software development of the SUPRENUM project (the German supercomputer project) many parallel grid-oriented algorithms are being programmed, especially multigrid and computational fluid dynamics codes. As the communication tasks are similar, a central SUPRENUM communications library with subroutines covering all communication requirements of the applications programs has been created. By implementing versions of the library for the Intel iPSC hypercube and the planned SUPRENUM machine, full portability of the applications software has been achieved.

1. Introduction

The German supercomputer project SUPRENUM (a German acronym of supercomputer for numerical applications) consists of three parts:

1. A distributed memory machine with a hierarchical, bus-oriented architecture. Each node processor is designed to have its own memory of 8 Mbytes size and a peak floating-point speed of some 20 MFLOPS (Millions of FLoating-point Operations Per Second). These processors are grouped in clusters of 16 units each. The SUPRENUM-1 machine will contain 16 clusters, so its overall memory size will be 2 Gbytes and its speed will be in the gigaflops range.
2. System software and programming languages. The host computer will run UNIX System-V, for the nodes special operating system software is under development. The programming languages FORTRAN and Modula-2 are extended by special constructs for message passing.
3. Applications software. Apart from basic software (e.g., a linear algebra package), applications programs in the field of partial differential equations are going to be written, many of them based on multigrid methods. The most important class of application programs handles computational fluid dynamic problems.

In the development of applications software, portability has always been of great importance. For sequential machines FORTRAN 77 provides a satisfactory base for the development of portable numerical software. While this is not always the case, a program that complies strictly to the programming language definition should execute correctly on most computers with valid compilers.

In the field of parallel computing, the situation is totally different. Although parallel computers usually support a dialect of FORTRAN 77, this is not sufficient to achieve portability. The concepts of communication between processes and/or synchronization vary widely, and no standard concept has arisen so far. Even if those tasks are performed by special subroutines (as is the case on the Intel iPSC which allows the user program to be free of non-standard constructs), portability is not achieved because the function of these subroutines is not standardized. Other parallel machines support

*Permanent Address: Gesellschaft für Mathematik und Datenverarbeitung mbH, Postfach 1240, 5205 St. Augustin, Federal Republic of Germany.

FORTRAN compilers with special language extensions for parallelization. In this case it is impossible to write parallel programs in standard FORTRAN 77.

In the SUPRENUM project this unsatisfactory situation led to the practice of writing applications programs in a modular way, with communication and arithmetic being performed in strictly separated subroutines. Implementation on another parallel machine thus limits rewriting to the communications subprograms. However, the range of applications programs planned for the SUPRENUM project would require a prohibitively large number of communications subroutines to be written.

One SUPRENUM subproject is the development of multigrid and computational fluid dynamics software. In this area many similar communications tasks occur in whole classes of applications programs, and these tasks can be performed by subroutines of a communications library. In the definition of these library routines, care has to be taken so that they are not designed to the special requirements of only one application. This obviously would unnecessarily restrict the range of use. Furthermore, it is important that the library routines cover all the required communications tasks of the applications programs under consideration; otherwise the user would again have to resort to non-standard constructs.

The most important aspect of the communications library is the minimization of redundant coding. Since many communications tasks are common to several applications classes, the provision of a library eliminates the duplication of these modules. Moreover, this approach reduces the total number of communications modules over the whole project, so the implementation of the library on other parallel machines becomes a more manageable task. As soon as the library is installed on different computers, all the applications programs can be ported to the new machine.

In the same way, optimization of the communications routines is facilitated. If, for example, another strategy in performing global operations turns out to be advantageous, this affects only the corresponding library subroutine rather than all applications program routines where global operations are executed.

At present, the SUPRENUM communications library has been implemented on the Intel iPSC hypercube and on the prototype of the SUPRENUM supercomputer. Thus the development of SUPRENUM application programs can be now be performed on the Intel hypercube, without any changes being necessary later.

As a first test of the library, a demonstration multigrid code for the Poisson equation on a rectangle with Dirichlet boundary conditions has been written with the communication done exclusively by library routines. The program has proved to be fully portable between the hypercube and SUPRENUM prototype, thus confirming the performance of the library.

The SUPRENUM communications library is, of course, not the first subroutine library for interprocess communication on parallel machines. The philosophy of this library, however, is that it is written on a fairly high level rather than simply exchanging (say) a send operation by a more portable construct. Instead, more complex tasks are performed by the library routines, facilitating the optimization of the underlying algorithm on the machine used. In fact, the Intel and SUPRENUM versions of many subroutines are very different; only the user interface remains unchanged.

The definition of the library is a dynamic process; additions are made as new requirements become apparent. So far, all communications requirements of multigrid programs and similar grid-related problems in two and three dimensions on rectangular or cuboid domains are covered. Corresponding routines are implemented for the SUPRENUM machine and the Intel hypercube. Often, for computational fluid dynamics applications, block-structured domains are used. Library routines for these more general domains have been defined but not yet implemented.

Section 2 discusses the conventions that must be satisfied by the user in allocating data. In Section 3, the communication tasks performed by the library subroutines are described in more detail. For the actual usage of the single routines, refer to the comment header of the library routines source codes in the Appendix.

2. Requirements for the User Program by the Communications Routines

The arithmetic subroutines written by the user and the library communications routines work on the same data allocated by the user, usually in his node main program. For the communications routines to work correctly, certain constraints must be satisfied by the user in making these allocations. This section is a preliminary approach to explain these conventions; a later publication will give some examples.

2.1. Grid Functions

Storage allocation for a distributed grid function is not as straightforward as in a serial code. Within the range of application of the library, parallelization is done by domain splitting; that is, the rectangular domain is subdivided into smaller rectangles (with some overlap), and each subdomain is assigned to a different process.

In the following discussion, we assume that the domain is two dimensional. Generalization to three dimensions is straightforward.

The domain decomposition leads to a "logical process grid," i.e., a template of processes and their neighborhood relations, in which the real assignment of these processes to real processors is not considered. In fact, for the applications program to be portable between different parallel machines, this point of view is the only one possible to the user, as the mapping of the processes onto the real processors is highly machine dependent. This mapping is done by special library routines, which are optimized to the underlying architecture. In this framework, each process is supplied with a logical process grid position (i,j), i and j being the x- and y-coordinates, respectively. The only information about the process location used in the application program is this logical process grid position.

It is further assumed that all the node programs are copies of the same code, the only difference being that they act on different parts of the domain. This is not as severe a restriction as it might appear, since the node program may contain branches, which depend on the location in the process grid.

The library routines must support a variable number of grid functions in the user program. Moreover, some communications routines act on all the grid functions simultaneously. Since a FORTRAN argument list must have a fixed number of entries, this is possible only by passing one address to the routine, together with a pointer vector containing the offsets of the single grid functions relative to that address. The easiest way for the user to do that is by allocating a single work vector

```
WORK(NDWORK)
```

in his main program, and then allocating all the N grid functions within this vector, the start addresses defined by

```
NSTART(1), NSTART(2), ... , NSTART(N).
```

Passing the grid functions to a communications routine then simply requires the arguments

```
WORK, NSTART, N.
```

A convention for addressing individual components of a distributed grid function is required to facilitate many data manipulations. The following technique has proven to be convenient and has therefore been adopted in the communications library as well as in the applications software developed in the SUPRENUM project.

Assume that a grid function is to be distributed over NX by NY processes. Each of the processes then gets assigned an inner portion of the global index range (1:NPX , 1:NPY), (where NPX and NPY are the global number of grid points in the x- and y-direction, respectively).

The node main program calculates its own inner index range from the global point grid dimensions and the position of the given process in the process grid, resulting in the inner index range

(IL:IX , JL:JX)

being assigned to that process. (The term "inner" is used to refer to a grid location that is updated by a particular processor; these values must provide a strict partition of the domain.) In addition to these inner points, some overlap with adjacent processes is needed. Thus, the actual storage allocation has to be somewhat larger, resulting in, say, the index bounds

(IDL:IDX , JDL:JDX)

with

IDL .le. IL .le. IX .le. IDX
JDL .le. JL .le. JX .le. JDX.

The space occupied by the grid function in the given process is thus

(IDX-IDL+1) (JDX-JDL+1).

If this refers to the first grid function, then the storage allocation should be

NSTART(1) = 1
NSTART(2) = NSTART(1) + (IDX-IDL+1) (JDX-JDL+1).

In a two-dimensional grid application one would like to have the grid functions declared as two-dimensional fields. At first sight, this conflicts with the storage allocation method given above, with all grid functions being sections in a one-dimensional vector.

In a subroutine called by the main program sketched below, this two-dimensional allocation for, say, the second grid function can be done nicely as follows:

Main program:

```
REAL WORK(10000)
        .
        .
        .
CALL SUB (WORK(NSTART(2)), IDL, IDX, JDL, JDX, IL, IX, JL, JX)
        .
        .
        .
END
```

Subroutine SUB:

```
SUBROUTINE SUB (U, IDL, IDX, JDL, JDX, IL, IX, JL, JX)
REAL U(IDL:IDX, JCL:JDX)
        .
        .
        .
END
```

A nice property of this data structure is that point indices in the subroutines are always global, i.e., refer to the position in the global point grid. Not only is that logically more appealing than usage of local indices, but also the code can be adapted to shared memory or to sequential machines more easily this way.

2.2. Tag Conventions

Some library routines take as an input argument a tag for messages to be passed by the routine to other processes. This gives the user the possibility to keep the messages for different tasks from interfering with each other. This is, for example, important if the user calls routine GLOPS twice in each process for performing different global operations. The messages going around can be sorted out correctly only if GLOPS is called with different tags.

Other routines taking tags as input are the exchange routines and the programs for sending subarrays to distant processes.

Some library routines, however, use tags themselves. To avoid conflicts with the user tags, the following subdivision of the available tag space is assumed:

00000 .le. tag .le. 27999: user-defined tags
28000 .le. tag .le. 28999: tags used by library routines
29000 .le. tag .le. 29999: tags used by mapping routines (SUPRENUM version only)

3. Overview of the Tasks Performed by the Communications Library

As a general rule, the communications library routines minimize the number of messages. This is important on machines that have high message startup times as compared to their communication bandwidth. For both the Intel iPSC hypercube under IOS (Intel operating system) and the SUPRENUM machine, this is the case.

On the hypercube, a message is passed as a vector (buffer) to the system-provided send routine. Thus one message can have only one data type. However, often data of different types are to be sent at the same time. To avoid multiple messages in this situation, an integer vector overlaying sections of different types is sent.

In the following subsections, an overview over the currently available subroutines is given, ordered by fields of application. A detailed description of each library routine can be found in the Appendix.

3.1. Creation of Node Processes

The subroutines CRGR2D (CReate a GRid of 2 Dimensions) and CRGR3D create node processes in a given 2D or 3D grid structure, respectively. The user choses the number of processes and the boundary condition (periodic or not) in each coordinate direction. The processes are organized in a grid and a tree structure. Each node process gets a message containing information about its position in the grid and tree, together with global information passed by the user to CRGR2D/CRGR3D.

The routines GRID2D and GRID3D are the node counterparts to the host routines CRGR2D and CRGR3D, respectively. They are to be called at the beginning of the node program. GRID2D/GRID3D receive the data sent by the host routines, initialize some common blocks used by later communication routines, and (on return) pass the global user data to the node program.

3.2. Global Operation over All Node Processes

In many applications global variables are calculated (e.g., residuals in relaxation routines). In a parallel program these variables have to be combined over all processes by application of an arithmetic operator (in the above example, "+" is used for L2-Norm, "max" is used for supremum norm, etc.). The node routine GLOPS performs this operations. Different operations on different data types (INTEGER, REAL) can be done simultaneously, thus saving the number of messages passed between processors.

Optionally, the results can be broadcasted to all participating node processes and/or to the host process.

If the results are to be sent to the host process, they are received there with routine GLOPH.

3.3. Exchange of Grid Function Values among Inner Boundaries

In iterative parallel algorithms on grids, data have to be exchanged along inner boundaries (i.e., boundaries between subgrids assigned to different processes). It is therefore necessary that, in a given process, not only the points belonging to that process are stored, but additionally all the points in an overlap region around the process subgrid. In the data exchange, variable values in this outer region are updated by the values calculated in those processes where the corresponding points are assigned.

In the current implementation of the library, this task has been split up into sending and receiving the exchange data. This allows the user to insert arithmetic tasks between update 'sends' and their corresponding 'receives', in order to optimize the performance of the code. The 'send' routines are SUPDT2 (two dimensions, grid-centered variables only), SUPSG2 (two dimensions, grid-centered or staggered variables), and SUPDT3 (three dimensions, grid-centered variables only). The corresponding 'receive' routines are RUPDT2, RUPSG2, and RUPDT3, respectively.

The user chooses the number of grid functions to be exchanged simultaneously, the width of the overlap region, the location of each function relative to the grid (only for SUPSG2 and RUPSG2), the direction of the exchange (e.g., all directions, only horizontally, only upwards), and a "coloring scheme" for the exchange data (e.g., exchange only "red" or only "black" points, exchange only every second row).

3.4. Sending a Subarray to a Specific Process

Usually, grid-related algorithms need only communication routines as described in Sections 3.1 through 3.3. However, there are special cases not covered by those routines, where a one-, two-, or three-dimensional array section has to be sent to some distant process. As a final resort for those cases, routines for sending and receiving integer and real subarrays have been implemented.

Subroutines SENDIA and RECIA send and receive, respectively, integer arrays. The user specifies the dimensioning of the subarray to be sent and the destination in terms of the logical process grid. He need not bother about the mapping of the processes onto the actual processors, as this is machine dependent and hidden to the user.

The counterparts to the above routines for real arrays are SENDRA and RECRA.

All these routines can be used in both the host process and the node processes in exactly the same way. If there are differences in the definition of the send and receive constructs between host and node (as, for example, in the case of the Intel hypercube), they are taken care of by the communications routines.

3.5. Node-triggered Abort of Distributed Application

If in a distributed application a user node program detects an error that renders a continuation of the run useless, the application should be stopped. On many parallel computers, however, only the host process can do this. This is why a subroutine INTRPT has been included in the library, which may be called from any node process. It first writes out a user-supplied error message and then informs the host program about the error. The next communications routine called on the host immediately returns with an error code of −1, telling the user program that an abort should be performed.

3.6. Redistribution of Grid Functions, Change of Process Grid

In a simple grid problem the subdivision of the grid onto the processes is done once and for all before the algorithm starts. In multigrid algorithms, however, a special problem arises on the coarsest grids. As the grids get coarser, the number of inner points per process decreases, and on the very coarsest grids some processes may not even contain any points at all. As a consequence on these grids the arithmetic work is overwhelmingly dominated by the time spent for communication. The messages are very short but numerous, which is especially problematical with communications systems with high startup time.

It turns out* that in such cases it may be better to collect the coarse grid problem onto a smaller process grid. Although some, or even most, processes are idle on the coarsest grids, this can be beneficial because of the reduction of the communications overhead. The best strategy may even be a sequence of such "agglomeration" steps, in which the process grid gets coarser in tandem with the point grid.

The communications library routine AGGLM2 does this agglomeration in two-dimensional applications. It is called by the user program in an (nx,ny) process configuration created by the routines CRGR2D/GRID2D. The user specifies the desired new process grid numbers (nxnew,nynew) with nxnew.le.nx, nynew.le.ny. AGGLM2 reorganizes the process grid, telling some processes to become inactive and redefining the environment of the processes that remain active. This is all hidden to the

*A. Krechel, private communication.

user. One may simply call routines such as SUPDT2/RUPDT2 after agglomeration and these routines are executed in the agglomerated grid environment. Along with the environment redefinition, the grid functions have to be collected from the fine process grid to the resulting coarser one. Many options as to the number of grid functions and additional data to be agglomerated without fitting in the framework of grid functions are offered by AGGLM2.

AGGLM2 may be called more than once, so that a series of agglomeration levels is supported. This allows the user to switch the process grids in a flexible way and to tune his algorithm precisely.

The inverse operation of agglomeration is also performed by AGGLM2. In this case the user specifies (−1,−1) for (nxnew,nynew). A de-agglomeration step always refers to the last agglomeration step.

3.7. Future Prospects

As mentioned above, the library is still in a state of development. In the near future, a three-dimensional agglomeration routine in analogy to the existing two-dimensional one will be included.

Another point is the input of initial data for grid functions from the host process to the nodes and the output of solution vectors from the nodes to the host. This can be programmed by using the routines SENDIA/RECIA and SENDRA/RECRA, and actually it has been done this way in the example multigrid code. However, more efficient solutions to these tasks are under consideration.

All the routines implemented so far work only on regular two- or three-dimensional grids. The corresponding routines for block-structured domains (as needed, for example, for SUPRENUM computational fluid dynamics applications) will be implemented soon.

4. Implementation of the SUPRENUM Communications Library, Hypercube Version

This section contains information on how to implement the communications library on an Intel hypercube. At Argonne National Laboratory, the library has been implemented on both a four-dimensional and five-dimensional hypercube. In addition, the bsim simulator together with the communications library has been installed on the VAX in Argonne's Mathematics and Computer Science Division. The library has been tested with the multigrid demonstration program on both the Intel hypercube and the simulator.

4.1. Name Conventions

4.1.1. Subroutines to Be Called by the User

The names of all subroutines to be called by the user are abbreviations related to the task they perform (e.g., SENDIA stands for SEND Integer Array). This is done to facilitate memorization of these names.

4.1.2. Subroutines Called Only by Library Routines

Ancillary subroutines called only by library routines all are named cmlnnn, with cml standing for CoMmunications Library and nnn being a three-digit number between 000 and 499. The user should avoid using any global names beginning with cml followed by a three-digit number.

4.1.3. Common Blocks Used by the Communications Library

Common blocks used by the library for internal purposes are named cmlnnn, with nnn being a three-digit number between 599 and 999. This range is further subdivided:

cm1500 through cm1599: common blocks containing information, which under certain circumstances could be useful to the user.

cm1600 through cm1999: common blocks for internal purposes only.

4.2. Parameters in the Source Codes

In the source codes of the library routines, some constants have been set by parameter statements (their meanings are declared in the corresponding subroutine headers). Some parameters are likely to need change on other Intel hypercubes:

iluser = length of a user interface integer in bytes. All integer arguments of library routines are declared with this length.

actual value: iluser = 4

irclen = length of a user interface real in bytes. All real arguments of library routines are declared with this length.

actual value: irclen = 8

intlen = length of a standard integer variable in bytes.

actual value: intlen = 4

maxdim = maximum dimension of cube (i.e., the maximum dimension provided by the hardware).

actual value: maxdim = 6

np = maximum number of processors.

actual value: np = 64

4.3. Non-Standard Declaration Statements

In the library subroutines, declaration statements of the form

```
integer*(iluser) .....
```

have been used, although this is not covered by the standard FORTRAN definition. However, this is convenient, as switching from one variable length to another can be done by just changing the parameter statement. Many compilers accept this construct. If the compiler used at a given site does not take these declarations, they can be replaced by a global change editor command. Replace

integer*(iluser)	by	integer*4	(if iluser=4),
real*(irelen)	by	real*8	(if irelen=8).

All other declarations are standard conforming.

4.4. Dummy Subroutines for Avoiding Unresolved Externals

Some of the library routines can be called from either host or node programs. As the system-supplied communications routines on the Intel hypercube are not the same on host and node, the routines must contain both host and node versions. The node routine is never called if the program is executed on the host, and vice versa. However, there are problems with linking a program, as the node routines are not in the host runtime library, and vice versa, resulting in unresolved external references. To avoid this, simply build a dummy library containing the programs

```
subroutine recvmsg
end
subroutine sendmsg
end
subroutine recvw
end
subroutine sendvw
end
subroutine probe
end
```

and put it into both host and node link steps as the last library.

4.5. Sample Makefile for Both Host and Node Executables

The following makefile gives an example for building the host and node code using the communications library on the Intel hypercube. Of course, file names have to be adjusted to the names used at a specific site.

```
.SUFFIXES:  .f .o

.f.o :
            -rmfort-b -h -e $<
#
#  note:  'b and h' switches are for contiguous huge array support
#         'e' is to list errors
#
#  These are commonly used switches.  See the RM FORTRAN User's Guide
#  for more details on all compiler switches.
#
#  Also, the '-' preceding 'rmfort' tells make to ignore errors and
#  warnings.  This is used because RM FORTRAN produces a warning when
#  different argument types are used on multiple subroutine calls, often
#  done on the send and receive message and type parameters.  Make treats
#  all warnings and errors the same, as fatal errors, and aborts when any
#  are found.  The '-' can sometimes cause strange results also.  If your
#  program actually had an error the '-' will cause it to be ignored and
#  make will link your .o which can produce an invalid executable.  To
#  be safe, you should always examine the output of make before you
#  execute the resulting file.

both : host node

help :    @echo "make both or make - makes both the host and node processes"
          @echo "make host   - makes the host process"
          @echo "make node   - makes the node process"
          @echo "make clean  - cleans up"

host:     host.o
          cc -Ml -o host  host.o \
          /usr/local/communic/library.a \
          /lib/Llibf.a \
          /usr/ipsc/lib/rmfhost.a \
          /usr/local/communic/dummy/library.a

node:     node.o
          ld -Ml -o node \
          /lib/Lseg.o \
          /usr/ipsc/lib/Lcrtn0.o \
          node.o \
          /usr/local/communic/library.a \
          /usr/ipsc/lib/Llibfnode.a \
          /usr/ipsc/lib/Llibcnode.a \
          /usr/local/communic/dummy/library.a

clean:    -rm *.o host node
```

4.6. Sample Makefile for Using the Library with the Simulator

On the VAX in Argonne's Mathematics and Computer Science Division, the Intel simulator bsim has been installed together with the communications library. The following makefile can be used for building the host and node executables for a simulation run and for starting the simulator:

```
.SILENT:
.SUFFIXES:  .f.o
.f.o :
        echo Compiling $<
        f77 -c $<

both: host node
        /user2/intel.sim/bsim

host:  host.o
        f77 -o host host.o /user2/hempel/communic/library.a \
                  /user2/intel.sim/bsimlib.a

node:  node.o
        f77 -o node node.o /user2/hempel/communic/library.a \
                  /user2/intel.sim/bsimlib.a

clean: rm host node host.o node.o LOGFILE
        ls -l

help:  echo make host:" "create new load module for the host process
        echo make node:" "create new load module for node process
        echo make both:" "create new load module for both host and node processes
        echo make clean:" "remove files allocated during simulations
```

4.7. Files on the Distribution Diskette

The diskette contains the FORTRAN source codes of all the communications routines (Intel iPSC version) available at the moment of the latest update. In addition, the file "tutorial" contains this text, "makefile" contains the sample makefile for the hypercube simulator (listed under Section 4.6), and "makefile.ipsc" is the sample makefile for the Intel iPSC hypercube (described in Section 4.5).

Acknowledgments

Many people have contributed to the work described in this paper. To all of them I express my gratitude, especially to H. Ritzdorf for writing the SUPRENUM version of the library routines, to A. Schueller for his collaboration in writing the parallel multigrid demonstration program, to A. Krechel and T. Niestegge for their performance tests of multigrid algorithms on the Intel iPSC, and to C. A. Thole for the definition of communication routines for block-structured domains.

I am most grateful to C. P. Thompson of Argonne National Laboratory for many valuable suggestions to the definition and implementation of the library as well as for his assistance in the preparation of the English version of the manuscript. I also thank J. M. Beumer and G. Pieper for typesetting and editing the final version of the paper, and the Mathematics and Computer Science Division of Argonne National Laboratory for the support of my work on the hypercubes at Argonne.

Appendix

Function and Calling Sequence of All Communications Subroutines

```
      subroutine agglm2 (nxnew, nynew, inew, jnew,
     +                   array, n, nstart, iworkl, iworkx,
     +                   idl, idx, jdl, jdx,
     +                   il, ix, jl, jx,
     +                   re, lenre, idimre, in, lenin, idimin, iorder,
     +                   repeat,
     +                   error)
c*********************************************************************
c
c  node routine
c  ------------
c
c  Agglomeration routine for two dimensional process grids.
c  This routine maps a (nx,ny) logical process grid to a new grid of
c  (nxnew,nynew) processes.
c  On output, (inew,jnew) is the position of the current process in this
c  grid. If (inew,jnew) = (0,0), then the current process is not active
c  in the new process grid.
c
c  Both the old and new process grid dimensions (i.e. (nx,ny), and
c  (nxnew,nynew) must be powers of two (including 1). The new mapping is
c  then done onto a nearest neighbor grid, and all messages during the
c  agglomeration are nearest neighbor messages.
c
c  On input the user specifies the grid functions on the current grid
c  level (arguments array, n, nstart) and their associated dimensions
c  (idl - jdx) and the inner portions (il - jx). For a process active on
c  the new grid these arguments are output arguments, too, giving
c  the corresponding values for the agglomerated grid functions.
c
c  Additionally, the user may pass two vectors to agglm2, i.e. the
c  real vector re and the integer vector in, containing any information
c  to be agglomerated together with the grid functions without having
c  that data structure.
c
c  agglm2 also perfoims the inverse operations, i.e. splitting up the
c  grid functions, sending them back to the original processes and
c  reactivating those processes. In this case, the user sets nxnew
c  and nynew to -1. (See the definition of the argument list below.)
c
c  Agglomeration may be performed in more than one stage. A scatter
c  operation (nxnew=nynew=-1) always refers to the last agglomeration.
c
c  Note: if the user only wants to establish the new process configu-
c        ration without merging any data from the agglomerated pro-
c        cesses, he can achieve this by calling AGGLM2 with n=0,
c        lenre=0 and lenin=0 on all processes. In this case no messages
c        are exchanged, which leads to a very quick execution.
c
c        On the other hand, if data are to be merged along the spanning
c        trees, the user must not choose n=lenre=lenin=0 in one of the
c        participating processes, because this would create a break in
c        the tree. This will result in process dead-lock.
c
c        If grid functions are to be agglomerated, this cannot happen,
c        because n has to be the same on all processes.
c
c        Caution: If, however, n=0, then the user has to be aware of
c        the problem mentioned above, as the values of lenre and lenin
c        may vary from process to process, and include zero values.
c
c
c
c  The calling sequence:
c
c  nxnew:  Number of processes in x-direction in target process grid
```

```
c
c  nynew:  Number of processes in y-direction in target process grid
c          special case nxnew=nynew=-1: inverse operation to last
c                                       agglomeration
c
c  inew:   x-coordinate of own process in process grid after
c          agglomeration
c
c  jnew:   y-coordinate of own process in process grid after
c          agglomeration
c
c          If inew=jnew=0 on output, then this process is inactive on
c          the new grid.
c
c  array  :  Real work vector containing grid functions
c
c  n      :  Number of grid functions to be exchanged (n.ge.0)
c
c  nstart :  integer vector of dimension n with start addresses of grid
c            functions in vector "array",
c            i.e. grid functions i starts at array(nstart(i))
c
c  iworkl: start index of free space in vector "array". This space
c          is used by agglm2 for allocation of the agglomerated grid
c          functions.
c
c  iworkx: end index of free space in vector "array", (cf. iworkl)
c
c          Note that iworkx-iworkl must be large enough to contain the
c          appropriate portions of the agglomerated grids.
c
c  idl-jdx:  dimension of grid functions
c
c  il - jx:  inner point limits of grid functions
c
c            The meaning of idl-jdx and il-ix may become clearer in
c            the following picture:
c
c          jdx  --------------------------
c              | All points stored in the |   if array(nstart(i)) is
c              | region of this process   |   passed to a subroutine,
c          jx  |      --------------      |   the corresponding grid
c              |     | inner points |     |   function may be dimen-
c              |     | (updated in  |     |   sioned there as
c              |     | this process)|     |
c          jl  |      --------------      |   u(idl:idx, jdl:jdx)
c              |                          |
c              |                          |
c          jdl  --------------------------
c             idl    il              ix   idx
c
c            It is assumed that each point belongs to one and only one
c            process, i.e. there is no overlap of inner points. So, the
c            il-value of a given process is just the ix-value of its
c            predecessor in x-direction plus one (etc.).
c
c  re:     real vector, containing additional information to be
c          agglomerated together with the grid functions
c
c  lenre:  number of entries in re to be included in agglomeration
c
c  idimre: dimension of vector re in calling program. In a process
c          which remains active after agglomeration, enough space must
c          be available in vector re for storage of all the collected
c          vector portions.
c
```

```
c  in:      integer vector, containing additional information to be
c           agglomerated together with the grid functions
c
c  lenin:   number of entries in in to be included in agglomeration
c
c  idimin:  dimension of vector in in calling program. In a process
c           which remains active after agglomeration, enough space must
c           be available in vector in for storage of all the collected
c           vector portions.
c
c  iorder:  if lenre.gt.0 or lenin.gt.0, iorder defines the ordering of
c           the vector portions in "re" and "in" after agglomeration.
c
c           Consider the case where a set of processes:
c           ((I,J) with I in (i1,i2), J in (j1,j2)) are mapped to a
c           process INEW, JNEW. Then the data are mapped consecutively
c           into the vectors re and in on the new process in the order
c           of increasing I and J.
c
c           If iorder=1: storage is ordered first with I then J;
c           If iorder=2: storage is ordered first with J then I.
c
c  repeat:  logical, indicating whether agglomeration with the same
c           values of nx, ny, nxnew, nynew, and the same data structure
c           regarding grid functions and vectors "in" and "re" has been
c           done earlier. In this case, essential parts of storage
c           management and save area copying can be skipped.
c           repeat=.true.: the same agglomeration was performed before
c           repeat=.false.: this is an agglomeration call in a new
c                           situation
c
c  error:   return code: =0: no error
c                        =1: maximum message length exceeded
c                        =2: Difference of iworkx and iworkl too small
c                        =3: dimension of vector "in" too small
c                        =4: dimension of vector "re" too small
c                        =5: invalid nxnew or nynew
c                        =6: n greater than at corresponding
c                            agglomeration
c                        =7: iorder other than at corresponding
c                            agglomeration
c                        =-1: nxnew=nx and nynew=ny (nothing to be done)
c
c
c  Input arguments: nxnew, nynew, array, n, nstart, iworkl, iworkx,
c                   idl, idx, jdl, jdx, il, ix, jl, jx, re, lenre
c                   idimre, in, lenin, idimin, iorder, repeat
c
c  Output arguments: nxnew, nynew, inew, jnew, array, n, nstart,
c                    iworkl, iworkx, idl, idx, jdl, jdx, il, ix, jl, jx,
c                    re, lenre, in, lenin, error
c
c  Types of arguments:
c
c  real*(irelen)     : array, re
c
c  Integer*(iluser)  : nxnew, nynew, inew, jnew, n, nstart, iworkl,
c                      iworkx, idl, idx, jdl, jdx, il, ix, jl, jx,
c                      lenre, idimre, in, lenin, idimin, iorder, error
c
c  logical           : repeat
c
c  AGGLM2 uses information stored in common blocks cml501, cml504,
c  cml505 and cml507, set by subroutine GRID2D.
c
c-----------------------------------------------------------------------
```

```
c
c  Version date: 05/20/1987   (version 1.1)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine crgr2d (nx, ny, slave, period, re, lenre, in, lenin,
     +                   error)
c***********************************************************************
c
c  host routine
c  ------------
c
c  crgr2d creates nx*ny processes whose path name is contained in the
c  character variable slave. It links the processes in both a 2-d grid
c  and a binary tree and sends to each process informations on its
c  neighbor processes and its own position in grid and tree.
c
c  The routine crgr2d is the counterpart to grid2d in the node
c  processes.
c
c  The calling sequence:
c
c  nx     :  Number of processes in x direction (nx.ge.1)
c  ny     :  Number of processes in y direction (ny.ge.1)
c  slave  :  Path name of node program
c  period :  Periodic boundary conditions? (vector, length=2)
c            period(1) = .T.: period. BC in x direction
c                      = .F.: no periodic BC in x direction
c            period(2) = .T.: period. BC in y direction
c                      = .F.: no periodic BC in y direction
c  re     :  Real vector, containing global parameters to be sent to
c            all processes
c  lenre  :  length of re (lenre.ge.0)
c  in     :  Integer vector, containing global parameters to be sent to
c            all processes
c  lenin  :  length of in (lenin.ge.0)
c  error  :  Return code: =0: no error
c                         =1: nx*ny greater than maximum number
c                             of processors
c                         =2: dimension of buffer cml601 to small
c                         =3: message longer than allowed (parameter
c                                                          maxmes)
c
c  All arguments except error are input arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : nx, ny, lenre, in, lenin, error
c  Real*(irelen)    : re
c  logical          : period
c  character*(*)    : slave
c
c
c  After return from CRGR2D, the following informations are stored in
c  common blocks to be used by other communication routines later on:
c
c  Block cml501:
c
c  ci:           communication channel
c  pid:          process id: always set to 1
c
c  Block cml504:
c
c  myno:         node id of current task
c
c  Block cml505:
c
c  idim:         dimension number of current application (2 or 3)
c  nx:           number of processes in x-direction
c  ny:           number of processes in y-direction
```

```
c  nz:           fictituous number of processes in z-direction, set to 1
c  i:            own x-index in logical process grid
c  j:            own y-index in logical process grid
c  k:            own z-index in logical process grid, set to 1
c  period(3):    for the three coordinate directions:
c                period(i) = .true. if periodic boundary conditions in
c                                    that direction, = .false. otherwise
c
c  Block cml507:
c
c  procn(nx,ny): integer array:
c  procn(i,j):   node id of logical process i,j, 1 <= i <= nx
c                                                1 <= j <= ny
c  Block cml507 is extended with dummy entries to a fixed length of
c  intlen*np bytes, with np=maximum number of processors and intlen=
c  length of an integer variable in bytes.
c
c-----------------------------------------------------------------------
c
c  Version date: 02/19/1987   (version 1.1)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine crgr3d (nx, ny, nz, slave, period, re, lenre, in,
     +                   lenin, error)
c***********************************************************************
c
c  host routine
c  ------------
c
c  crgr3d creates nx*ny*nz processes whose path name is contained in the
c  character variable slave. It links the processes in both a 3-d grid
c  and a binary tree and sends to each process informations on its
c  neighbor processes and its own position in grid and tree.
c
c  The routine crgr3d is the counterpart to grid3d in the node
c  processes.
c
c  The calling sequence:
c
c  nx     :  Number of processes in x direction (nx.ge.1)
c  ny     :  Number of processes in y direction (ny.ge.1)
c  nz     :  Number of processes in z direction (nz.ge.1)
c  slave  :  Path name of node program
c  period :  Periodic boundary conditions?  (vector, length=3)
c            period(1) = .T.: period. BC in x direction
c                      = .F.: no periodic BC in x direction
c            period(2) = .T.: period. BC in y direction
c                      = .F.: no periodic BC in y direction
c            period(3) = .T.: period. BC in z direction
c                      = .F.: no periodic BC in z direction
c  re     :  Real vector, containing global parameters to be sent to
c            all processes
c  lenre  :  length of re (lenre.ge.0)
c  in     :  Integer vector, containing global parameters to be sent to
c            all processes
c  lenin  :  length of in (lenin.ge.0)
c  error  :  Return code: =0: no error
c                         =1: nx*ny*nz greater than maximum number
c                             of processors
c                         =2: dimension of buffer cml601 to small
c                         =3: message longer than allowed (parameter
c                                                          maxmes)
c
c  All arguments except error are input arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : nx, ny, nz, lenre, in, lenin, error
c  Real*(irelen)    : re
c  logical          : period
c  character*(*)    : slave
c
c
c  After return from CRGR2D, the following informations are stored in
c  common blocks to be used by other communication routines later on:
c
c  Block cml501:
c
c  ci:           communication channel
c  pid:          process id: always set to 1
c
c  Block cml504:
c
c  myno:         node id of current task
c
c  Block cml505:
c
```

```
c  idim:          dimension number of current application (2 or 3)
c  nx:            number of processes in x-direction
c  ny:            number of processes in y-direction
c  nz:            number of processes in z-direction
c  i:             own x-index in logical process grid
c  j:             own y-index in logical process grid
c  k:             own z-index in logical process grid
c  period(3):     for the three coordinate directions:
c                 period(i) = .true. if periodic boundary conditions in
c                                    that direction, = .false. otherwise
c
c  Block cml507:
c
c  procn(nx,ny,nz), integer array:
c  procn(i,j,k):   node id of logical process i,j,k,  1 <= i <= nx
c                                                     1 <= j <= ny
c                                                     1 <= k <= nz
c  Block cml507 is extended with dummy entries to a fixed length of
c  intlen*np bytes, with np=maximum number of processors and intlen=
c  length of an integer variable in bytes.
c
c-----------------------------------------------------------------------
c
c  Version date: 05/07/1987   (version 1.2)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine gloph (re, lenre, in, lenin, tag, error)
c***********************************************************************
c
c  host routine
c  ------------
c
c  gloph is used to receive results of global operations on real and/or
c  integer data over all node processes. It is the counterpart to the
c  node routine glops, which collects and processes the data (using a
c  tree structure) and sends them to the host process if requested by
c  the user.
c
c  The calling sequence:
c
c  re         Real vector, output
c  lenre      Number of received entries in vector re, output
c  in         Integer vector, output
c  lenin      Number of received entries in vector in, output
c  tag        Identifier of message, input
c  error      Return code: =0: no error
c                          =1: received message longer than buffer cml601
c                         =-1: abort message issued by some node process,
c                              whole distributed application should be
c                              stopped.
c
c  Data types of arguments:
c  Real*(irelen)    : re
c  Integer*(iluser): lenre, in, lenin, tag, error
c
c  Remark: Be sure that vectors re and in are dimensioned big enough in
c          the calling program. gloph cannot check their length and
c          copies as many elements into these vectors as it receives
c          from the node processes.
c
c  GLOPH uses data stored in common block cml501 by routine CRGR2D or
c  CRGR3D, respectively.
c
c-----------------------------------------------------------------------
c
c  Version date: 05/07/1987    (version 1.1)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine glops (re, reop, lenre, in, inop, lenin, back, host,
     +                  tag, error)
c***********************************************************************
c
c  node routine
c  ------------
c
c  glops is used to perform global operations elementwise on real vector
c  re and  integer vector in along a binary tree structure. The global
c  results can be sent to the host process and/or back to all node
c  processes.
c
c  The calling sequence:
c
c  re     :  Real vector, input (and output, if back=.T.)
c  reop   :  Character vector, input. reop(i) specifies, which operation
c            is to be performed on re(i):
c              reop(i) = '+': addition
c              reop(i) = '*': multiplication
c              reop(i) = 'H': Maximum (highest value)
c              reop(i) = 'L': Minimum (lowest value)
c  lenre  :  Dimension of re and reop (lenre.ge.0)
c  in     :  Integer vector, input (and output, if back=.T.)
c  inop   :  Character vector, input. inop(i) specifies, which operation
c            is to be performed on in(i):
c              inop(i) = '+': addition
c              inop(i) = '*': multiplication
c              inop(i) = 'H': Maximum (highest value)
c              inop(i) = 'L': Minimum (lowest value)
c  lenin  :  Dimension of in and inop (lenin.ge.0)
c  back   :  Option: back=.T.: send results back to all node processes
c                        =.F.: don't send results back
c  host      Option: host=.T.: send results to host process
c                        =.F.: don't send results to host
c  tag    :  Identifier of message
c  error  :  Return code: =0: no error
c                         =1: dimension of buffer ibuf too small
c                         =2: message longer than allowed (maxmes)
c                         =3: received message has wrong length
c                         =4: lenre or lenin negative
c                         =5: message received from wrong process
c                         =6: invalid option in vector inop
c                         =7: invalid option in vector reop
c                         =8: back and host are both .false.
c
c  Data types of arguments:
c  Real*(irelen)    : re
c  Integer*(iluser): lenre, in, lenin, tag, error
c  Logical          : back, host
c  Character*1      : reop, inop
c
c  All arguments except error are input arguments.
c
c  GLOPS uses information contained in common blocks cm1501 and cm1502,
c  set by routine GRID2D or GRID3D, respectively.
c
c-----------------------------------------------------------------------
c
c  Version date: 05/07/1986   (version 1.1)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c***********************************************************************
```

```
      subroutine grid2d (nx, ny, i, j, re, lenre, in, lenin, error)
c*********************************************************************
c
c  node routine
c  ------------
c
c  grid2d is the counterpart to the host routine crgr2d. It receives
c  the initial informations sent by that routine to the node processes
c  and passes them to the calling program.
c
c  The calling sequence:
c
c  nx     :  Number of processes in x-direction
c  ny     :  Number of processes in y-direction
c  i, j   :  Own position in nx*ny - process grid
c  re     :  Real vector, containing global parameters received from
c            host
c  lenre  :  length of re
c  in     :  Integer vector, containing global parameters received from
c            host
c  lenin  :  length of in
c  error  :  Return code: =0: no error
c                         =1: received message longer than buffer
c                         =2: received message has wrong length
c                         =3: received message is too short
c                         =4: buffer cml600 is too short
c
c  All arguments are output arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : nx, ny, i, j, lenre, in, lenin, error
c  Real*(irelen)    : re
c
c
c  After return from GRID2D, the following informations are stored in
c  common blocks to be used by other communication routines later on:
c
c  Block cml500:
c
c  next      process id's of neighbors:                Integer array,
c                                                      dim(2,2)
c            next (1,1) = lower x neighbor
c            next (1,2) = upper x neighbor
c            next (2,1) = lower y neighbor
c            next (2,2) = upper y neighbor
c
c  Block cml501:
c
c  ci:       communication channel
c  pid:      process id: always set to 1
c
c  Block cml502:
c
c  master :  process id of the host process            Integer
c  tree   :  Information on father and sons in binary
c            tree:                                     Integer, Vector
c            tree(1) = father
c            tree(2), tree(3) = sons
c
c  Block cml504:
c
c  myno:     node id of current task
c
c  Block cml505:
```

```
c
c  idim    :    dimension number of current application (2 or 3)
c  nx      :    number of processes in x-direction        Integer
c  ny      :    number of processes in y-direction        Integer
c  nz      :    fictituous 3. dimension: set to 1         Integer
c  i       :    own x-index in logical process grid
c  j       :    own y-index in logical process grid
c  k       :    own z-index in logical process grid, set to 1
c  period(3):  for the three coordinate directions:
c              period(i) = .true. if periodic boundary conditions in
c                                 that direction, = .false. otherwise
c              period(3) is not used in two-dimensional applications.
c
c  Block cml507:
c
c  procn(nx,ny): integer array:
c  procn(i,j):   node id of logical process i,j, 1 <= i <= nx
c                                                1 <= j <= ny
c  An empty process id is coded by -1.
c
c  Block cml507 is extended with dummy entries to a fixed length of
c  intlen*np bytes, with np=maximum number of processors and intlen=
c  length of an integer variable in bytes.
c
c-----------------------------------------------------------------------
c
c  Version date: 05/07/1987   (version 1.2)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine grid3d (nx, ny, nz, i, j, k, re, lenre, in, lenin,
     +                   error)
c*********************************************************************
c
c  node routine
c  ------------
c
c  grid3d ist the counterpart to the host routine crgr3d. It receives
c  the initial informations sent by that routine to the node processes
c  and passes them to the calling program.
c
c  The calling sequence:
c
c  nx      :  Number of processes in x-direction
c  ny      :  Number of processes in y-direction
c  nz      :  Number of processes in z-direction
c  i, j, k:  Own position in nx*ny*nz - process grid
c  re      :  Real vector, containing global parameters received from
c             host
c  lenre   :  length of re
c  in      :  Integer vector, containing global parameters received from
c             host
c  lenin   :  length of in
c  error   :  Return code: =0: no error
c                          =1: received message longer than buffer
c                          =2: received message has wrong length
c                          =3: received message is too short
c                          =4: buffer cml600 is too short
c
c  All arguments are output arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : nx, ny, nz, i, j, k, lenre, in, lenin, error
c  Real*(irelen)    : re
c
c
c  After return from CRGR3D, the following informations are stored in
c  common blocks to be used by other communication routines later on:
c
c  Block cml501:
c
c  ci:       communication channel
c  pid:      process id: always set to 1
c
c  Block cml502:
c
c  master :  process id of the host process            Integer
c  tree   :  Information on father and sons in binary
c            tree:                                    Integer, Vector
c            tree(1) = father
c            tree(2), tree(3) = sons
c
c  Block cml503:
c
c  next      process id's of neighbors:               Integer array,
c                                                     dim(3,2)
c            next (1,1) = lower x neighbor
c            next (1,2) = upper x neighbor
c            next (2,1) = lower y neighbor
c            next (2,2) = upper y neighbor
c            next (3,1) = lower z neighbor
c            next (3,2) = upper z neighbor
c
c  Block cml504:
```

```
c
c  myno:     node id of current task
c
c  Block cml505:
c
c  idim      :    dimension number of current application (2 or 3)
c  nx        :    number of processes in x-direction      Integer
c  ny        :    number of processes in y-direction      Integer
c  nz        :    number of processes in z-direction      Integer
c  i         :    own x-index in logical process grid     Integer
c  j         :    own y-index in logical process grid     Integer
c  k         :    own z-index in logical process grid, set to 1, Integer
c  period(3):    for the three coordinate directions:    Logical
c                period(i) = .true. if periodic boundary conditions in
c                                   that direction, = .false. otherwise
c
c  Block cml507:
c
c  procn  :  process id's of entire process grid:    Integer array,
c            procn(i,j,k) = Process id of process    dim(nx,ny,nz)
c                           with logical process
c                           grid coordinates i,j,k
c  An empty process id is coded by -1.
c
c  Block cml507 is extended with dummy entries to a fixed length of
c  intlen*np bytes, with np=maximum number of processors and intlen=
c  length of an integer variable in bytes.
c
c-----------------------------------------------------------------------
c
c  Version date: 05/07/1987   (version 1.2)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine intrpt (string)
c**********************************************************************
c
c  node routine
c  ------------
c
c  intrpt is a node routine which gives the user the possibility to stop
c  a distributed application. If intrpt is called on a node, the user
c  supplied message contained in the character string "string" is
c  written to the syslog file and a message is sent to the host process
c  which in turn issues an error code -1 the next time a communication
c  routine is called there. It is then up to the user to stop the host
c  process immediately or to print out some additional information etc.
c  first.
c
c  The calling sequence:
c
c  string :  Abort message to be put on the syslog file (input)
c
c  Type of argument:
c
c  Character*(*) : string
c
c----------------------------------------------------------------------
c
c  Version date: 10/28/1986   (version 1.0)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c**********************************************************************
```

```
      subroutine recia (intarr, idl, idx, jdl, jdx, kdl, kdx,
     +                  imin, imax, jmin, jmax, kmin, kmax,
     +                  tag, error)
c*********************************************************************
c
c  host/node routine
c  -----------------
c
c  recia receives a subarray of the integer array intarr from another
c  process. The user specifies the tag of the incoming message.
c
c  The calling sequence:
c
c  intarr : Integer array, dimension: intarr (idl:idx, jdl:jdx, kdl:kdx)
c  idl    : lowest first index of intarr (see intarr)
c  idx    : highest first index of intarr (see intarr)
c  jdl    : lowest second index of intarr (see intarr)
c  jdx    : highest second index of intarr (see intarr)
c  kdl    : lowest third index of intarr (see intarr)
c  kdx    : highest third index of intarr (see intarr)
c  imin   : lowest first index of the subarray to be received
c  imax   : highest first index of the received subarray
c  jmin   : lowest second index of the subarray to be received
c  jmax   : highest second index of the received subarray
c  kmin   : lowest third index of the subarray to be received
c  kmax   : highest third index of the received subarray
c  tag    : identifier of message
c  error  : return code: =0: no error
c                        =1: received message too short
c                        =2: received message has wrong length
c                        =3: too few message segments received
c                       =-1: abort message issued by some node process,
c                            whole distributed application should be
c                            stopped.
c
c  Input : idl, idx, jdl, jdx, kdl, kdx, imin, jmin, kmin, tag.
c  Output: intarr(imin:imax, jmin:jmax, kmin:kmax), imax, jmax, kmax,
c          error.
c
c
c  Types of arguments:
c
c  Integer*(iluser): intarr, idl, idx, jdl, jdx, kdl, kdx,
c                    imin, imax, jmin, jmax, kmin, kmax, tag, error
c
c  RECIA uses information stored in common blocks cml501 and cml504,
c  set by one of the following routines: CRGR2D, CRGR3D, GRID2D, GRID3D.
c
c---------------------------------------------------------------------
c
c  Version date: 11/06/1986   (version 1.0)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c*********************************************************************
```

```
      subroutine recra (array, idl, idx, jdl, jdx, kdl, kdx,
     +                  imin, imax, jmin, jmax, kmin, kmax,
     +                  tag, error)
c*********************************************************************
c
c  host/node routine
c  -----------------
c
c  recra receives a subarray of the real array "array" from another
c  process. The user specifies the tag of the incoming message.
c
c  The calling sequence:
c
c  array  : Real array, dimension: array (idl:idx, jdl:jdx, kdl:kdx)
c  idl    : lowest first index of array (see array)
c  idx    : highest first index of array (see array)
c  jdl    : lowest second index of array (see array)
c  jdx    : highest second index of array (see array)
c  kdl    : lowest third index of array (see array)
c  kdx    : highest third index of array (see array)
c  imin   : lowest first index of the subarray to be received
c  imax   : highest first index of the received subarray
c  jmin   : lowest second index of the subarray to be received
c  jmax   : highest second index of the received subarray
c  kmin   : lowest third index of the subarray to be received
c  kmax   : highest third index of the received subarray
c  tag    : identifier of message
c  error  : return code: =0: no error
c                        =1: received message too short
c                        =2: received message has wrong length
c                        =3: too few message segments received
c                       =-1: abort message issued by some node process,
c                            whole distributed application should be
c                            stopped.
c
c  Input : idl, idx, jdl, jdx, kdl, kdx, imin, jmin, kmin, tag.
c  Output: array(imin:imax, jmin:jmax, kmin:kmax), imax, jmax, kmax,
c          error.
c
c
c  Types of arguments:
c
c  real*(irelen)    : array
c
c  Integer*(iluser): idl, idx, jdl, jdx, kdl, kdx,
c                    imin, imax, jmin, jmax, kmin, kmax, tag, error
c
c  RECRA uses information stored in common blocks cml501 and cml504,
c  set by one of the following routines: CRGR2D, CRGR3D, GRID2D, GRID3D.
c
c---------------------------------------------------------------------
c
c  Version date: 05/07/1987   (version 1.1)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c*********************************************************************
```

```
      subroutine recvwh (ci, type, buf, len, cnt, node, pid, error)
c*********************************************************************
c
c  host routine
c  ------------
c
c  recvwh simulates the communication routine recvw in the node
c  processes. The argument list is identical to that one of the node
c  routine, except for the argument "error" which has been included
c  because of possible buffer overflows.
c
c
c  The buffer length "len" must be a multiple of parameter intlen,
c  otherwise the last few bytes of a message may be missing.
c
c  On output, "error" gives the following return code:
c              error = 0: no error
c              error = 1: received message longer than buffer,
c                         parameter meslen too small
c              error = 2: too many messages in buffer, parameter nmes
c                         too small
c              error = 3: no more space in segmented buffer bufs,
c                         parameter nseg too small
c
c  If in some node process the communication subroutine "abort" is
c  called, that routine sends a dummy message with a special tag to
c  the host. As soon as recvwh gets such a message, it stops operation
c  and returns to the calling program with the return code set to
c              error = -1.
c  So the host user program is informed that the whole application
c  has to be stopped.
c
c---------------------------------------------------------------------
c
c  Version date: 05/09/1987   (version 1.1)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c*********************************************************************
```

```
      subroutine rupdt2 (array, n, nstart, idl, idx, jdl, jdx,
     +                   il, ix, jl, jx, color, width, dest, tag, error)
c***********************************************************************
c
c  node routine
c  ------------
c
c  rupdt2 receives the values of grid functions at points in the overlap
c  zones on inner process interfaces from the neighbor processes. The
c  width of this zone as well as the direction in which data are to
c  be exchanged can be chosen by the user. More than one grid function
c  can be received simultaneously.
c
c  The calling sequence:
c
c  array  :  Real work vector containing grid functions
c  n      :  Number of grid functions to be exchanged
c  nstart :  integer vector of dimension n with start addresses of grid
c            functions in vector "array",
c            i.e. grid functions i starts at array(nstart(i))
c  idl-jdx:  dimension of grid functions
c  il - jx:  inner point limits of grid functions
c            (For more details regarding idl - jx, see explanatory part
c            of routine supdt2)
c  color  :  selection code for class of exchange points:
c            color=0: all points in overlap zone
c            color=1: only points with i odd
c            color=2: only points with i even
c            color=3: only points with j odd
c            color=4: only points with j even
c            color=5: only points with i+j odd
c            color=6: only points with i+j even
c  width  :  width of overlap zone
c  dest   :  from which direction have data to be received?
c            dest(1,1)=.true.: from upper x-neighbor
c            dest(1,2)=.true.: from lower x-neighbor
c            dest(2,1)=.true.: from upper y-neighbor
c            dest(2,2)=.true.: from lower y-neighbor
c            Attention: the meaning of the "dest" entries is reversed
c            as compared to the definition in the supdt2 routine. The
c            reason is that sending of a message in a given direction
c            corresponds to receiving a message from the opposite
c            direction.
c  tag    :  identifier of message: tag, tag+1, tag+2 and tag+3 can
c            be used by rupdt2.
c            Be sure to use the same tag for the corresponding calls
c            of routines supdt2 and rupdt2.
c  error  :  Return code: =0: no error
c                         =2: received message has wrong length
c                         =3: invalid color selected
c                         =4: inner region narrower than exchange width
c
c  All arguments except array and error are input arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : n, nstart, idl, idx, jdl, jdx, il, ix, jl, jx,
c                     color, width, tag, error
c  Real*(irelen)    : array
c  logical          : dest
c
c-----------------------------------------------------------------------
c
c  Version date: 05/07/1987   (version 1.1)
c
```

```
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c*****************************************************************
```

```
      subroutine rupdt3 (array, n, nstart, idl, idx, jdl, jdx, kdl, kdx,
     +                   il, ix, jl, jx, kl, kx, color, width, dest,
     +                   tag, error)
c***********************************************************************
c
c  node routine
c  ------------
c
c  rupdt3 receives the values of grid functions at points in the overlap
c  zones on inner process interfaces from the neighbor processes. The
c  width of this zone as well as the direction in which data are to
c  be exchanged can be chosen by the user. More than one grid function
c  can be received simultaneously.
c
c  The calling sequence:
c
c  array  :  Real work vector containing grid functions
c  n      :  Number of grid functions to be exchanged
c  nstart :  integer vector of dimension n with start addresses of grid
c            functions in vector "array",
c            i.e. grid functions i starts at array(nstart(i))
c  idl-kdx:  dimension of grid functions
c  il - kx:  inner point limits of grid functions
c            (For details about idl-kx, see the explanatory part of the
c            two-dimensional routine supdt2.)
c  color  :  selection code for class of exchange points:
c            color=0:  all points in overlap zone
c            color=1:  only points with i+j+k odd
c            color=2:  only points with i+j+k even
c            color=3:  only points with i+j odd
c            color=4:  only points with i+j even
c            color=5:  only points with i+k odd
c            color=6:  only points with i+k even
c            color=7:  only points with j+k odd
c            color=8:  only points with j+k even
c            color=9:  only points with i odd
c            color=10: only points with i even
c            color=11: only points with j odd
c            color=12: only points with j even
c            color=13: only points with k odd
c            color=14: only points with k even
c  width  :  width of overlap zone
c  dest   :  from which direction have data to be received?
c            dest(1,1)=.true.: from upper x-neighbor
c            dest(1,2)=.true.: from lower x-neighbor
c            dest(2,1)=.true.: from upper y-neighbor
c            dest(2,2)=.true.: from lower y-neighbor
c            dest(3,1)=.true.: from upper z-neighbor
c            dest(3,2)=.true.: from lower z-neighbor
c            Attention: the meaning of the "dest" entries is reversed
c            as compared to the definition in the supdt3 routine. The
c            reason is that sending of a message in a given direction
c            corresponds to receiving a message from the opposite
c            direction.
c  tag    :  identifier of message: tag, tag+1, tag+2, tag+3, tag+4
c            and tag+5 can be used by rupdt3. Moreover, if the amount
c            of data to be sent to a neighbor process exceeds the
c            maximum message length, then for the second messages
c            tags in the range tag+6 through tag+11 are used, and so on.
c  error  :  Return code: =0: no error
c                         =1: invalid color selected
c                         =2: inner region narrower than exchange width
c
c  All arguments except array and error are input arguments.
c
c
```

```
c  Types of arguments:
c
c  Integer*(iluser) : n, nstart, idl, idx, jdl, jdx, kdl, kdx,
c                     il, ix, jl, jx, kl, kx, color, width, tag, error
c  Real*(irelen)    : array
c  logical          : dest
c
c-----------------------------------------------------------------------
c
c  Version date: 05/07/1987   (version 1.1)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine rupsg2 (array, n, nstart, nloc,
     +                   idl, idx, jdl, jdx, il, ix, jl, jx,
     +                   color, width, dest, tag, error)
c***********************************************************************
c
c  node routine
c  ------------
c
c  Exchange-Receive routine for staggered grids
c
c  rupsg2 receives the values of grid functions at points in the overlap
c  zones on inner process interfaces from the neighbor processes. The
c  width of this zone as well as the direction from which data are to
c  be received can be chosen by the user. More than one grid function
c  can be handled simultaneously. For each grid function the location in
c  the corresponding control volume can be selected. rupsg2 is the
c  counterpart to the send routine supsg2.
c
c  The calling sequence:
c
c  array  :  Real work vector containing grid functions
c  n      :  Number of grid functions to be exchanged
c  nstart :  integer vector of dimension n with start addresses of grid
c            functions in vector "array",
c            i.e. grid functions i starts at array(nstart(i))
c  nloc   :  integer vector of dimension n, giving for each grid
c            function the location of the variable relative to the
c            corresponding control volume according to the following
c            scheme:
c
c                      7        8       9
c                        +------+-----+
c                        |             |
c                        |             |
c                      4 +      +      + 6
c                        |        5    |
c                        |             |
c                        +------+-----+
c                      1        2       3
c
c  idl-jdx:  dimension of grid functions
c  il - jx:  inner point limits of grid functions
c            (For details about idl - jx, see the explanatory part of
c            subroutine supsg2)
c  color     selection code for class of exchange cells:
c            color=0: all cells in overlap zone
c            color=1: only cells with i odd
c            color=2: only cells with i even
c            color=3: only cells with j odd
c            color=4: only cells with j even
c            color=5: only cells with i+j odd
c            color=6: only cells with i+j even
c            for each cell only the associated variables are exchanged.
c  width  :  width of overlap zone:
c            the absolute value of "width" gives the width of the
c            overlap zone, measured in cells. If width is positive,
c            gridfunctions laying on the process border are included
c            in the exchange, if it is negative, only "true" inner
c            points are received.
c  dest   :  from which direction have data to be received?
c            dest(1,1)=.true.: from upper x-neighbor
c            dest(1,2)=.true.: from lower x-neighbor
c            dest(2,1)=.true.: from upper y-neighbor
c            dest(2,2)=.true.: from lower y-neighbor
c            Attention: the meaning of the "dest" entries is reversed
c            as compared to the definition in the supsg2 routine. The
```

```
c            reason is that sending of a message in a given direction
c            corresponds to receiving a message from the opposite
c            direction.
c  tag    :  identifier of message: tag, tag+1, tag+2 and tag+3 can
c            be used by rupsg2.
c            Be sure to use the same tag for the corresponding calls
c            of routines supsg2 and rupsg2.
c  error  :  Return code: =0: no error
c                          =2: received message has wrong length
c                          =3: invalid color selected
c                          =4: inner region narrower than exchange width
c
c  All arguments except array and error are input arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : n, nstart, nloc, idl, idx, jdl, jdx,
c                     il, ix, jl, jx, color, width, tag, error
c  Real*(irelen)    : array
c  logical          : dest
c
c-----------------------------------------------------------------------
c
c  Version date: 05/07/1987   (version 1.1)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine sendia (intarr, idl, idx, jdl, jdx, kdl, kdx,
     +                   imin, imax, jmin, jmax, kmin, kmax,
     +                   idest, jdest, kdest, tag, error)
c*********************************************************************
c
c  host/node routine
c  -----------------
c
c  sendia sends a subarray of the integer array intarr to another
c  process. The user specifies the destination of the message and its
c  tag.
c
c  If the message length exceeds 16 Kbytes, it has to be split into
c  smaller segments. Because these segments have to be sent with unique
c  tags, sendia in this case uses the identifier tag, tag+1, tag+2, and
c  so on. The user shouldn't use these tags for his own purposes, in
c  order to avoid confusion.
c
c  The calling sequence:
c
c  intarr : Integer array, dimension: intarr (idl:idx, jdl:jdx, kdl:kdx)
c  idl    : lowest first index of intarr (see intarr)
c  idx    : highest first index of intarr (see intarr)
c  jdl    : lowest second index of intarr (see intarr)
c  jdx    : highest second index of intarr (see intarr)
c  kdl    : lowest third index of intarr (see intarr)
c  kdx    : highest third index of intarr (see intarr)
c  imin   : lowest first index of the subarray to be sent
c  imax   : highest first index of the subarray to be sent
c  jmin   : lowest second index of the subarray to be sent
c  jmax   : highest second index of the subarray to be sent
c  kmin   : lowest third index of the subarray to be sent
c  kmax   : highest third index of the subarray to be sent
c  idest,
c  jdest,
c  kdest  : logical process grid coordinates of destination process.
c           In case of a twodimensional problem, argument kdest is not
c           used.
c           If idest=jdest=kdest=0 or idest=jdest=0, respectively, then
c           the message is sent to the host process.
c  tag    : identifier of message
c  error  : return code: =0: no error
c                        =1: argument idest out of range
c                        =2: argument jdest out of range
c                        =3: argument kdest out of range
c
c  All arguments except error are input arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : intarr, idl, idx, jdl, jdx, kdl, kdx,
c                     imin, imax, jmin, jmax, kmin, kmax,
c                     idest, jdest, kdest, tag, error
c
c  RECIA uses information stored in common blocks cml501, cml504,
c  cml505 and cml507 set by one of the following routines: CRGR2D,
c  CRGR3D, GRID2D, GRID3D.
c
c---------------------------------------------------------------------
c
c  Version date: 03/25/1987   (version 1.2)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
```

```
c          5205 St. Augustin
c          West Germany
c
c-----------------------------------------------------------------------
c
c  Changes against version 1.1:
c
c  In version 1.1, argument kdest was changed by SENDIA in twodimen-
c  sional applications. So it was an output argument in that case.
c  It now isn't changed by SENDIA in any case.
c
c***********************************************************************
```

```
      subroutine sendra (array, idl, idx, jdl, jdx, kdl, kdx,
     +                   imin, imax, jmin, jmax, kmin, kmax,
     +                   idest, jdest, kdest, tag, error)
c*********************************************************************
c
c  host/node routine
c  -----------------
c
c  sendra sends a subarray of the real array "array" to another
c  process. The user specifies the destination of the message and its
c  tag.
c
c  If the message length exceeds 16 Kbytes, it has to be split into
c  smaller segments. Because these segments have to be sent with unique
c  tags, sendra in this case uses the identifier tag, tag+1, tag+2, and
c  so on. The user shouldn't use these tags for his own purposes, in
c  order to avoid confusion.
c
c  The calling sequence:
c
c  array  : real array, dimension: array (idl:idx, jdl:jdx, kdl:kdx)
c  idl    : lowest first index of array (see array)
c  idx    : highest first index of array (see array)
c  jdl    : lowest second index of array (see array)
c  jdx    : highest second index of array (see array)
c  kdl    : lowest third index of array (see array)
c  kdx    : highest third index of array (see array)
c  imin   : lowest first index of the subarray to be sent
c  imax   : highest first index of the subarray to be sent
c  jmin   : lowest second index of the subarray to be sent
c  jmax   : highest second index of the subarray to be sent
c  kmin   : lowest third index of the subarray to be sent
c  kmax   : highest third index of the subarray to be sent
c  idest,
c  jdest,
c  kdest  : logical process grid coordinates of destination process.
c           In case of a twodimensional problem, argument kdest is not
c           used.
c           If idest=jdest=kdest=0 or idest=jdest=0, respectively, then
c           the message is sent to the host process.
c  tag    : identifier of message
c  error  : return code: =0: no error
c                        =1: argument idest out of range
c                        =2: argument jdest out of range
c                        =3: argument kdest out of range
c
c  All arguments except error are input arguments.
c
c
c  Types of arguments:
c
c  real*(irelen)     : array
c
c  Integer*(iluser)  : idl, idx, jdl, jdx, kdl, kdx,
c                      imin, imax, jmin, jmax, kmin, kmax,
c                      idest, jdest, kdest, tag, error
c
c  RECIA uses information stored in common blocks cml501, cml504,
c  cml505 and cml507, set by one of the following routines:
c  CRGR2D, CRGR3D, GRID2D, GRID3D.
c
c---------------------------------------------------------------------
c
c  Version date: 03/25/1987   (version 1.2)
c
c  Author: Rolf Hempel
```

```
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c-----------------------------------------------------------------------
c
c  Changes against version 1.1:
c
c  In version 1.1, argument kdest was changed by SENDRA in twodimen-
c  sional applications. So it was an output argument in that case.
c  It now isn't changed by SENDRA in any case.
c
c***********************************************************************
```

```
      subroutine supdt2 (array, n, nstart, idl, idx, jdl, jdx,
     +                   il, ix, jl, jx, color, width, dest, tag, error)
c*********************************************************************
c
c  node routine
c  ------------
c
c  supdt2 sends the values of grid functions at points in the overlap
c  zones on inner process interfaces to the neighbor processes. The
c  width of this zone as well as the direction in which data are to
c  be exchanged can be chosen by the user. More than one grid function
c  can be sent simultaneously.
c
c  The calling sequence:
c
c  array  :  Real work vector containing grid functions
c  n      :  Number of grid functions to be exchanged (n.ge.1)
c  nstart :  integer vector of dimension n with start addresses of grid
c            functions in vector "array",
c            i.e. grid functions i starts at array(nstart(i))
c  idl-jdx:  dimension of grid functions
c  il - jx:  inner point limits of grid functions
c            The meaning of idl-jdx and il-ix may become clearer in
c            the following picture:
c
c           jdx  --------------------------
c               | All points stored in the |   if array(nstart(i)) is
c               | region of this process   |   passed to a subroutine,
c           jx  |      --------------      |   the corresponding grid
c               |     | inner points |     |   function may be dimen-
c               |     | (updated in  |     |   sioned there as
c               |     | this process)|     |
c           jl  |      --------------      |   u(idl:idx, jdl:jdx)
c               |                          |
c               |                          |
c           jdl  --------------------------
c              idl    il              ix   idx
c
c             It is assumed that each point belongs to one and only one
c             process, i.e. there is no overlap of inner points. So, the
c             il-value of a given process is just the ix-value of its
c             predecessor in x-direction plus one.
c
c  color     selection code for class of exchange points:
c            color=0: all points in overlap zone
c            color=1: only points with i odd
c            color=2: only points with i even
c            color=3: only points with j odd
c            color=4: only points with j even
c            color=5: only points with i+j odd
c            color=6: only points with i+j even
c  width  :  width of overlap zone
c  dest   :  in which direction should data be transmitted?
c            dest(1,1)=.true.: to lower x-neighbor
c            dest(1,2)=.true.: to upper x-neighbor
c            dest(2,1)=.true.: to lower y-neighbor
c            dest(2,2)=.true.: to upper y-neighbor
c  tag    :  identifier of message: tag, tag+1, tag+2 and tag+3 can
c            be used by supdt2.
c            Be sure to use the same tag for the corresponding calls
c            of routines supdt2 and rupdt2.
c  error  :  Return code: =0: no error
c                         =1: maximum message length exceeded
c                         =2: buffer cml600 is too short
c                         =3: invalid color selected
c                         =4: inner region narrower than exchange width
```

```
c            In case of error=1 or error=2, there is only a remedy if
c            more than one grid function is handled at a time. In this
c            case, use the exchange routines for the grid functions
c            seperately. Two-dimensional exchange data are assumed
c            to fit into a single message, so they aren't split up.
c
c  All arguments except error are input arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : n, nstart, idl, idx, jdl, jdx, il, ix, jl, jx,
c                     color, width, tag, error
c  Real*(irelen)    : array
c  logical          : dest
c
c-----------------------------------------------------------------------
c
c  Version date: 01/20/1987   (version 1.0)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Daten erarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine supdt3 (array, n, nstart, idl, idx, jdl, jdx, kdl, kdx,
     +                   il, ix, jl, jx, kl, kx, color, width, dest,
     +                   tag, error)
c*****************************************************************************
c
c  node routine
c  ------------
c
c  supdt3 sends the values of grid functions at points in the overlap
c  zones on inner process interfaces to the neighbor processes. The
c  width of this zone as well as the direction in which data are to
c  be exchanged can be chosen by the user. More than one grid function
c  can be sent simultaneously.
c
c  The calling sequence:
c
c  array  :  Real work vector containing grid functions
c  n      :  Number of grid functions to be exchanged (n.ge.1)
c  nstart :  integer vector of dimension n with start addresses of grid
c            functions in vector "array",
c            i.e. grid functions i starts at array(nstart(i))
c  idl-kdx:  dimension of grid functions: u(idl:idx, jdl:jdx, kdl:kdx)
c  il - kx:  inner point limits of grid functions
c            (For details on idl - kx, see the explanatory part of
c            the two-dimensional counterpart, i.e. subroutine supdt2)
c  color  :  selection code for class of exchange points:
c            color=0:  all points in overlap zone
c            color=1:  only points with i+j+k odd
c            color=2:  only points with i+j+k even
c            color=3:  only points with i+j odd
c            color=4:  only points with i+j even
c            color=5:  only points with i+k odd
c            color=6:  only points with i+k even
c            color=7:  only points with j+k odd
c            color=8:  only points with j+k even
c            color=9:  only points with i odd
c            color=10: only points with i even
c            color=11: only points with j odd
c            color=12: only points with j even
c            color=13: only points with k odd
c            color=14: only points with k even
c  width  :  width of overlap zone
c  dest   :  in which direction should data be transmitted?
c            dest(1,1)=.true.: to lower x-neighbor
c            dest(1,2)=.true.: to upper x-neighbor
c            dest(2,1)=.true.: to lower y-neighbor
c            dest(2,2)=.true.: to upper y-neighbor
c            dest(3,1)=.true.: to lower z-neighbor
c            dest(3,2)=.true.: to upper z-neighbor
c  tag    :  identifier of message: tag, tag+1, tag+2, tag+3, tag+4
c            and tag+5 can be used by supdt3. Moreover, if the amount
c            of data to be sent to a neighbor process exceeds the
c            maximum message length, then for the second messages
c            tags in the range tag+6 through tag+11 are used, and so on.
c            Be sure to use the same tag for the corresponding calls
c            of routines supdt3 and rupdt3.
c  error  :  Return code: =0: no error
c                         =1: buffer cml600 is too short
c                         =2: invalid color selected
c                         =3: inner region narrower than exchange width
c            In case of error=1, there is only a remedy if more than one
c            grid function is handled at a time. In this case, use the
c            exchange routines for the grid functions seperately. For
c            three-dimensional exchange it is assumed that all data fit
c            into the communication buffer cml600.
c
```

```
c  All arguments except error are input arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : n, nstart, idl, idx, jdl, jdx, kdl, kdx,
c                     il, ix, jl, jx, kl, kx, color, width, tag, error
c  Real*(irelen)    : array
c  logical          : dest
c
c-----------------------------------------------------------------------
c
c  Version date: 01/20/1987   (version 1.0)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      subroutine supsg2 (array, n, nstart, nloc,
     +                   idl, idx, jdl, jdx, il, ix, jl, jx,
     +                   color, width, dest, tag, error)
c***********************************************************************
c
c  node routine
c  ------------
c
c  Exchange-Send routine for staggered grids
c
c  supsg2 sends the values of grid functions at points in the overlap
c  zones on inner process interfaces to the neighbor processes. The
c  width of this zone as well as the direction in which data are to
c  be exchanged can be chosen by the user. More than one grid function
c  can be sent simultaneously. For each grid function the location in
c  the corresponding mass control volume can be selected.
c
c  The calling sequence:
c
c  array  :  Real work vector containing grid functions
c  n      :  Number of grid functions to be exchanged (n.ge.1)
c  nstart :  integer vector of dimension n with start addresses of grid
c            functions in vector "array",
c            i.e. grid functions i starts at array(nstart(i))
c  nloc      integer vector, giving for each grid function the location
c            of the variable relative to the corresponding control
c            volume according to the following scheme:
c
c                      7        8       9
c                        +------+------+
c                        |             |
c                        |             |
c                      4 +      +      + 6
c                        |       5     |
c                        |             |
c                        +------+------+
c                      1        2       3
c
c  idl-jdx:  dimension of grid functions
c  il - jx:  inner point limits of grid functions
c            The meaning of idl-jdx and il-ix may become clearer in
c            the following picture:
c
c          jdx  --------------------------
c              | All cells stored in the  |   if array(nstart(i)) is
c              | region of this process   |   passed to a subroutine,
c          jx  |      --------------      |   the corresponding grid
c              |     | inner cells  |     |   function may be dimen-
c              |     | (updated in  |     |   sioned there as
c              |     | this process)|     |
c          jl  |      --------------      |   u(idl:idx, jdl:jdx)
c              |                          |
c              |                          |
c          jdl  --------------------------
c             idl    il              ix   idx
c
c            It is assumed that each cell belongs to one and only one
c            process, i.e. there is no overlap of inner cells. So, the
c            il-value of a given process is just the ix-value of its
c            predecessor in x-direction plus one.
c
c  color  :  selection code for class of exchange cells:
c            color=0: all cells in overlap zone
c            color=1: only cells with i odd
c            color=2: only cells with i even
c            color=3: only cells with j odd
```

```
c            color=4: only cells with j even
c            color=5: only cells with i+j odd
c            color=6: only cells with i+j even
c            for each cell only the associated variables are exchanged.
c  width   :  width of overlap zone:
c            the absolute value of "width" gives the width of the
c            overlap zone, measured in cells. If width is positive,
c            gridfunctions laying on the process border are included
c            in the exchange, if it is negative, only "true" inner
c            points are sent.
c  dest      in which direction should data be transmitted?
c            dest(1,1)=.true.: to lower x-neighbor
c            dest(1,2)=.true.: to upper x-neighbor
c            dest(2,1)=.true.: to lower y-neighbor
c            dest(2,2)=.true.: to upper y-neighbor
c  tag       identifier of message: tag, tag+1, tag+2 and tag+3 can
c            be used by supsg2.
c            Be sure to use the same tag for the corresponding calls
c            of routines supsg2 and rupsg2.
c  error   :  Return code: =0: no error
c                           =1: maximim message length exceeded
c                           =2: buffer cml600 is too short
c                           =3: invalid color selected
c                           =4: inner region narrower than exchange width
c            In case of error=1 or error=2, there is only a remedy if
c            more than one grid function is handled at a time. In this
c            case, use the exchange routines for the grid functions
c            seperately. Two-dimensional exchange data are assumed
c            to fit into a single message, so they aren't split up.
c
c  All arguments except error are input arguments.
c
c
c  Types of arguments:
c
c  Integer*(iluser) : n, nstart, nloc, idl, idx, jdl, jdx, il, ix, jl,
c                     jx, color, width, tag, error
c  Real*(irelen)    : array
c  logical          : dest
c
c-----------------------------------------------------------------------
c
c  Version date: 01/20/1987   (version 1.0)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c***********************************************************************
```

```
      logical function testtag (tag)
c*********************************************************************
c
c  host/node routine
c  -----------------
c
c  testtag tests whether a message with identifier tag is waiting for
c  reception in the current process. The result is:
c      testtag = .t., if at least one message with the given tag is
c                      waiting, and
c      testtag = .f., if no such message is waiting.
c  The function can be used in both host and node processes. If it is
c  used on the host, then:
c      1. subroutine recvwh has to be called at least once, and
c      2. it is only tested whether a message of the desired tag was
c         waiting at the last call of routine recvwh.
c  Subroutine recvwh is called by all routines of the communication
c  library which receive messages from the nodes.
c
c  The calling sequence:
c
c  tag :  Identifier of looked for message
c
c  Type of argument:
c
c  Integer*(iluser): tag
c
c---------------------------------------------------------------------
c
c  Version date: 10/31/1986   (version 1.0)
c
c  Author: Rolf Hempel
c          F1 / T
c          Gesellschaft fuer Mathematik und Datenverarbeitung, mbH
c          5205 St. Augustin
c          West Germany
c
c*********************************************************************
```

Distribution for ANL-87-23

Internal:

J. M. Beumer (2)
H. G. Kaper
A. B. Krisciunas
G. W. Pieper (37)
E. P. Steinberg

ANL Patent Department
ANL Contract File
ANL Libraries
TIS Files (5)

External:

DOE-TIC, for distribution per UC-32 (113)
Manager, Chicago Operations Office, DOE
Mathematics and Computer Science Division Review Committee:
  J. L. Bona, Pennsylvania State University
  T. L. Brown, University of Illinois, Urbana
  P. Concus, Lawrence Berkeley Laboratory
  S. Gerhart, Micro Electronics and Computer Technology Corp., Austin, TX
  H. B. Keller, California Institute of Technology
  J. A. Nohel, University of Wisconsin, Madison
  M. J. O'Donnell, University of Chicago
D. Austin, ER-DOE
G. Chesshire, Intel Corp. (5)
P. Frederickson, Los Alamos National Laboratory
R. Hempel, SUPRENUM Project, West Germany (40)
G. Michael, Lawrence Livermore Laboratory
D. Padua, University of Illinois, Urbana
E. van de Velde, California Institute of Technology