The Institute of Statistical Mathematics
Statistical Computing Technical Report

RSC-026 1995.12

(revised/translated 1999.10)

DEBB, a debugging bug, manual

(MPI-version)

Makio Ishiguro, The Institute of Statistical Mathematics

In debugging process it is desirable to have detailed output when and only when the bug is just coming. This is possible if

1. we can write commands in the output record of the program under concern, and
2. the program can read commands in the modified record in it's rerun, and
3. the program make suitable outputs obeying given commands.

DEBB package consists of subroutines summarized in Table 1 is developed to realize these functions.

Table 1 DEBB package

subroutine	function
bug	makes machine readable outputs and reads commands
sbuggle	reads commands
bugtrapv	checks unexpected changes of variables
bugtraph	checks unexpected discripancy among variables of different nodes in parallel proccesing
bugnet	observation of variables

A minimum tutorial course on the use of subroutines 'bug' and 'sbuggle' is given here. Some hints on debugging parallel computing program are also included. The full manual for DEBB routines will be given elsewhere.

1 An example

Simple.f shown in Fig.1 is a simple example. It might look too simple to be realistic. However the essence of the performance of the BUG subroutine is shown here, and you should be able to get some hint to handle difficult bugs.

The program shown in Fig. 1 is to compute

$$s = \sum_{i=1}^{500} \frac{1}{(1.1^i + 1)^3 - 1.1^{3i}} \tag{1}$$

```
      implicit real*8 (a-h,o-z)
      sum = 0.d0
      do 1 i = 1,500
         a = 1.1d0 ** i
         sum = sum + 1.d0 / (( a + 1.d0) ** 3 - a ** 3)
    1 continue
      write(6,*) 'sum =', sum
      stop
      end
```

Figure 1. Program Simple.f with bug

Compile the program simple.f and then run it. The record of the computation is shown in Fig. 2.

```
MI@sunmi% simple
sum = Infinity
 Note: the following IEEE floating-point arithmetic exceptions
 occurred and were never cleared; see ieee_flags(3M):
 Inexact;  Division by Zero;
```

Figure 2. A wrong result

sum becomes infinite because of some bug. Modify the program as shown in Fig. 3, re-compile and run it, then there comes a prompt

```
start: Bug ? (<Y>es/with <M>ap/<N>o bug)
```

Answer to this by entering 'y' with return key, then the result shown in Fig. 4 is obtained.

```
      implicit real*8 (a-h,o-z)
      sum = 0.d0
      call bug(0,0,0,0,'start',0,0.d0,message)
      do 1 i = 1,500
         a = 1.1d0 ** i
         sum = sum + 1.d0 / (( a + 1.d0) ** 3 - a ** 3)
         call bug(1,mod(i,20),0,0,'sum',i, sum , message )
         if(message .eq. 2) then
            write(6,*) i, ( a + 1.d0) ** 3 - a ** 3, a ** 3
         end if
    1 continue
      write(6,*) 'sum =', sum
      stop
      end
```

Figure 3. Modified program

```
MI@sunmi% simple
DEBugging Bug, version 4-MPI
start: Bug ? (<Y>es/with <M>ap/<N>o bug)
y
 DEBB started
 COM: 0:LOOK
 COM: 0:LETITGO      -10:
 MEM: 0: bug.map command list
 MEM: 0:  LOOK     LETITGO  MESSAGE  LEVEL    BACK     QUIT
 MEM: 0:  SKIP     DUMMY
 BUG: 0: 1:start   :   1:         0: 0.0000000000000000000000000000000000D+00:
 BUG: 0: 1:sum     :   2:        20: 0.8878842074243455950721681801951 49D+00:
 BUG: 0: 1:sum     :   3:        40: 0.9191050146953799737659096535935 53D+00:
 BUG: 0: 1:sum     :   4:        60: 0.9198520778648098250585007917834 45D+00:
 BUG: 0: 1:sum     :   5:        80: 0.9198687853874782938135012955171 99D+00:
 BUG: 0: 1:sum     :   6:       100: 0.9198691552066612375782028721005 19D+00:
 BUG: 0: 1:sum     :   7:       120: 0.9198691633799855882358542658039 37D+00:
 BUG: 0: 1:sum     :   8:       140: 0.9198691635605816818710422921867 58D+00:
 BUG: 0: 1:sum     :   9:       160: 0.9198691635645721564884524923400 02D+00:
 BUG: 0: 1:sum     :  10:       180: 0.9198691635646604192189101922849 66D+00:
 BUG: 0: 1:sum     :  11:       200: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  12:       220: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  13:       240: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  14:       260: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  15:       280: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  16:       300: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  17:       320: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  18:       340: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  19:       360: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  20:       380: 0.9198691635646621955757495925354 31D+00:
 BUG: 0: 1:sum     :  21:       400:Infinity                                :
 BUG: 0: 1:sum     :  22:       420:Infinity                                :
 BUG: 0: 1:sum     :  23:       440:Infinity                                :
 BUG: 0: 1:sum     :  24:       460:Infinity                                :
 BUG: 0: 1:sum     :  25:       480:Infinity                                :
 BUG: 0: 1:sum     :  26:       500:Infinity                                :
sum = Infinity
```

Figure 4. Result with BUG record lines.

This result reveals when the program start doing wrong. Prepare the 'bug.map' file shown in Fig. 5 by adding two comand lines to this record in Fig.4.

```
start: Bug ? (<Y>es/with <M>ap/<N>o bug)
y
 DEBB started
 COM: 0:LOOK
 COM: 0:LETITGO      -10:
 MEM: 0: bug.map command list
 MEM: 0:  LOOK     LETITGO  MESSAGE  LEVEL    BACK     QUIT
 MEM: 0:  SKIP     DUMMY
 BUG: 0: 1:start   :   1:         0: 0.00000000000000000000000000000000D+00:
 BUG: 0: 1:sum     :   2:        20: 0.88788420742434559507216818019514 9D+00:
 BUG: 0: 1:sum     :   3:        40: 0.91910501469537997376590965359355 3D+00:
 BUG: 0: 1:sum     :   4:        60: 0.91985207786480982505850079178344 5D+00:
 BUG: 0: 1:sum     :   5:        80: 0.91986878538747829381350129551719 9D+00:
 BUG: 0: 1:sum     :   6:       100: 0.91986915520666123757820287210051 9D+00:
 BUG: 0: 1:sum     :   7:       120: 0.91986916337998558823585426580393 7D+00:
 BUG: 0: 1:sum     :   8:       140: 0.91986916356058168187104229218675 8D+00:
 BUG: 0: 1:sum     :   9:       160: 0.91986916356457215648845249234000 2D+00:
 BUG: 0: 1:sum     :  10:       180: 0.91986916356466041921891019228496 6D+00:
 BUG: 0: 1:sum     :  11:       200: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     :  12:       220: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     :  13:       240: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     :  14:       260: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     :  15:       280: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     :  16:       300: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     :  17:       320: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     :  18:       340: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     :  19:       360: 0.91986916356466219557574959253543 1D+00:
 COM: 0:MESSAGE       2
 BUG: 0: 1:sum     :  20:       380: 0.91986916356466219557574959253543 1D+00:
 COM: 0:QUIT
 BUG: 0: 1:sum     :  21:       400:Infinity                                    :
 BUG: 0: 1:sum     :  22:       420:Infinity                                    :
 BUG: 0: 1:sum     :  23:       440:Infinity                                    :
 BUG: 0: 1:sum     :  24:       460:Infinity                                    :
 BUG: 0: 1:sum     :  25:       480:Infinity                                    :
 BUG: 0: 1:sum     :  26:       500:Infinity                                    :
sum = Infinity
```

Figure 5. 'bug.map'.

Run the program again and answer the prompt

```
start: Bug ? (<Y>es/with <M>ap/<N>o bug)
```

by entering 'm' this time. Then you get the result shown in Fig. 6.

```
MI@sunmi% simple

DEBugging Bug, version 4-MPI

start: Bug ? (<Y>es/with <M>ap/<N>o bug)
m
 DEBB started with a Map
 MEM: 0: bug.map command list
 MEM: 0:  LOOK     LETITGO  MESSAGE  LEVEL    BACK     QUIT
 MEM: 0:  SKIP     DUMMY
 COM: 0:LOOK
 COM: 0:LETITGO      -10:
 BUG: 0: 1:start   !   1:         0: 0.00000000000000000000000000000000D+00:
 BUG: 0: 1:sum     !   2:        20: 0.88788420742434559507216818019514 9D+00:
 BUG: 0: 1:sum     !   3:        40: 0.91910501469537997376590965359355 3D+00:
 BUG: 0: 1:sum     !   4:        60: 0.91985207786480982505850079178344 5D+00:
 BUG: 0: 1:sum     !   5:        80: 0.91986878538747829381350129551719 9D+00:
 BUG: 0: 1:sum     !   6:       100: 0.91986915520666123757820287210051 9D+00:
 BUG: 0: 1:sum     !   7:       120: 0.91986916337998558823585426580393 7D+00:
 BUG: 0: 1:sum     !   8:       140: 0.91986916356058168187104229218675 8D+00:
 BUG: 0: 1:sum     !   9:       160: 0.91986916356457215648845249234000 2D+00:
 BUG: 0: 1:sum     !  10:       180: 0.91986916356466041921891019228496 6D+00:
 BUG: 0: 1:sum     !  11:       200: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     !  12:       220: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     !  13:       240: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     !  14:       260: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     !  15:       280: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     !  16:       300: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     !  17:       320: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     !  18:       340: 0.91986916356466219557574959253543 1D+00:
 BUG: 0: 1:sum     !  19:       360: 0.91986916356466219557574959253543 1D+00:
 COM: 0:MESSAGE       2:
 BUG: 0: 1:sum     !  20:       380: 0.91986916356466219557574959253543 1D+00:
  380    6.0847228810955D+31    1.5404977927049D+47
  381    1.2169445762191D+32    2.0504025620902D+47
  382    1.6225927682921D+32    2.7290858101421D+47
  383    1.6225927682921D+32    3.6324132132992D+47
  384    2.4338891524382D+32    4.8347419869012D+47
  385    2.4338891524382D+32    6.4350415845655D+47
```

```
386    4.8677783048764D+32    8.5650403490566D+47
387  0.      1.1400068704594D+48
388    6.4903710731685D+32    1.5173491445815D+48
389  0.      2.0195917114380D+48
390    6.4903710731685D+32    2.6880765679240D+48
391  0.      3.5778299119068D+48
392  0.      4.7620916127480D+48
393  0.      6.3383439365675D+48
394  0.      8.4363357795714D+48
395  0.      1.1228762922610D+49
396  0.      1.4945483449993D+49
397  0.      1.9892438471941D+49
398  0.      2.6476835606154D+49
399  0.      3.5240668191790D+49
COM: 0:QUIT
BUG: 0: 1:sum     :  21:       400:Infinity                                    :
Note: the following IEEE floating-point arithmetic exceptions
occurred and were never cleared; see ieee_flags(3M):
Inexact;  Division by Zero;
```

Figure 6. Found bug

This result reveals that the error is caused by the wrong computation $(a+1)^3 - a^3 = 0$, namely the underflow.

This underflow can be avoided by changing the program so that it utilize the right hand side form of the equation

$$\frac{1}{(y+1)^3 - y^3} = \frac{1}{(y+1)^2 + (y+1)y + y^2}, \tag{2}$$

obtained by substituting

$$x = y + 1 \tag{3}$$

for the x of the equation

$$\frac{x-y}{x^3 - y^3} = \frac{1}{x^2 + xy + y^2} \tag{4}$$

2 Tutorial Course

1. Run the program:

```
      call bug (1,0,0,0,'start',0,0.d0,message)
      call bug (1,0,0,0,'tutorial',511,1999.d0,message)
      if( message .eq. 123 ) write(6,*) 'message is received'
      call sbuggle (jewel)
      if( jewel .eq. 456 ) write(6,*) 'message can be smuggled in'
      do 1 j = 1,10
         call bug (1,j,10,10,'hello',0,0.d0,message)
    1 continue
      call bug (1,j,10,10,'finish',0,0.d0,message)
```

   and you get nothing.

2. Change the first 1 in the first line of the program to 0 to get

```
      call bug (0,0,0,0,'start',0,0.d0,message)
```

3. Re-compile and run the modified program. Record the output in a file.

4. The prompt,

```
start: Bug ? (<Y>es / with <M>ap / <N>o bug )
```

   is given. Answer this by 'y' and see what happens.

   - The generic form of BUG call is

     ```
     call  bug ( Lid , J , J1 , J2 , Cid , Iid , Rid , Message )
     ```

     BUG routine becomes active only after a call with Lid = 0. BUG calls can be embedded in a program in the sleeping mode beforehand as the preparation for debugging in future.

   - The generic form of the output of BUG is as follows:

     ```
     BUG:id:Lid:Cid   :n: Iid:Rid
     ```

     In the MPI-parallel processing environment, the value of 'id' is the rank of the processor making the print-out. In a single processor machine, it is always 0. 'n' is the output order, others are values specified in BUG calls.

- You will find other bug calls are also activated and make their outputs.

5. Name the output record file 'bug.map'.
6. Edit 'bug.map'. There should be lines,

```
COM: 0:LOOK
COM: 0:LETITGO      -10:
MEM: 0: bug.map command list
MEM: 0:  LOOK     LETITGO  MESSAGE  LEVEL    BACK     QUIT
MEM: 0:  SKIP     DUMMY
BUG: 0: 1:start   :   1:          0:  .00000000000000000000000000000000D+00:
BUG: 0: 1:tutorial:   2:        511:  .19990000000000000000000000000000D+04:
BUG: 0: 1:hello   :   3:          0:  .00000000000000000000000000000000D+00:
BUG: 0: 1:finish  :   4:          0:  .00000000000000000000000000000000D+00:
```

Find the line,

```
BUG: 0: 1:hello   :   3:          0:  .00000000000000000000000000000000D+00:
```

and add a command line so that you have

```
COM: 0:QUIT
BUG: 0: 1:hello   :   3:          0:  .00000000000000000000000000000000D+00:
```

- The generic form of COMMAND is

  ```
  COM:id:CCCC    d
  ```

  'CCCC' is the command, '`d`' is an integer constant. There are commands which take no argument of '`d`' like 'QUIT' command.

- If your machine is a parallel computer, every machine will make print outs.
- If you are interested in the performance of the machine of 'rank' j, replace every '` COM: 0:`' by '` COM: j:`' in the bug.map.

7. Run the program and answer the prompt,

   ```
   start: Bug ? (<Y>es / with <M>ap / <N>o bug )
   ```

   by 'm' and see what happens.

- This operation is "to start bug with a map".
- Try another answer 'n' sometime.

8. When you start bug with the above map, the execution of the program stops after printing

```
BUG: 0: 1:start   !   1:          0:  .00000000000000000000000000000000D+00:
BUG: 0: 1:tutorial!   2:        511:  .19990000000000000000000000000000D+04:
COM: 0:QUIT
BUG: 0: 1:hello   :   3:          0:  .00000000000000000000000000000000D+00:
```

- BUG accepts 'QUIT' command.

9. Edit again 'bug.map' so that you have

```
BUG: 0: 1:start   :   1:          0:  .00000000000000000000000000000000D+00:
COM: 0:MESSAGE      123
BUG: 0: 1:tutorial:   2:        511:  .19990000000000000000000000000000D+04:
COM: 0:QUIT
BUG: 0: 1:hello   :   3:          0:  .00000000000000000000000000000000D+00:
BUG: 0: 1:finish  :   4:          0:  .00000000000000000000000000000000D+00:
```

run the program, start bug with the map, and you will have

```
BUG: 0: 1:start   !   1:          0:  .00000000000000000000000000000000D+00:
COM: 0:MESSAGE      123:
BUG: 0: 1:tutorial:   2:        511:  .19990000000000000000000000000000D+04:
message is received
COM: 0:QUIT
BUG: 0: 1:hello   :   3:          0:  .00000000000000000000000000000000D+00:
```

- BUG reads 'MESSAGE' and pass the information to the main routine through the last argument 'message'.

10. Edit again 'bug.map' and change the line

```
COM: 0:MESSAGE      123
```

to

```
COM: 0:MESSAGE      456
```

and run the program, start bug with the map, then you will have

```
BUG: 0: 1:start   !   1:         0:  .00000000000000000000000000000000D+00:
COM: 0:MESSAGE      456:
BUG: 0: 1:tutorial!   2:       511:  .19990000000000000000000000000000D+04:
message can be smuggled in
COM: 0:QUIT
BUG: 0: 1:hello   :   3:         0:  .00000000000000000000000000000000D+00:
```

- 'MESSAGE' can be smuggled in.

11. Edit the source file of the program to add a line

```
      call bug (2,j,7,8,'peekaboo',j,0.d0, message )
```

to get

```
       call bug (0,0,0,0,'start',0,0.d0,message)
       call bug (1,0,0,0,'tutorial',511,1999.d0,message)
       if( message .eq. 123 ) write(6,*) 'message is received'
       call sbuggle (jewel)
       if( jewel .eq. 456 ) write(6,*) 'message can be smuggled in'
       do 1 j = 1,10
          call bug (2,j,7,8,'peekaboo',j,0.d0, message )
          call bug (1,j,10,10,'hello',0,0.d0,message)
     1 continue
       call bug (1,j,10,10,'finish',0,0.d0,message)
```

then compile it and run the program, start bug with the map, then you will have

```
BUG: 0: 1:start   !   1:         0:  .00000000000000000000000000000000D+00:
COM: 0:MESSAGE      456:
BUG: 0: 1:tutorial!   2:       511:  .19990000000000000000000000000000D+04:
message can be smuggled in
COM: 0:QUIT
BUG: 0: 1:hello   :   3:         0:  .00000000000000000000000000000000D+00:
```

- Addition of a higher level BUG call with higher level 'Lid' value does not affect the performance of the program.

12. Edit again 'bug.map' so that you have

```
BUG: 0: 1:start   :   1:         0:  .00000000000000000000000000000000D+00:
COM: 0:LEVEL        2
COM: 0:MESSAGE      456
BUG: 0: 1:tutorial:   2:       511:  .19990000000000000000000000000000D+04:
COM: 0:QUIT
BUG: 0: 1:hello   :   3:         0:  .00000000000000000000000000000000D+00:
BUG: 0: 1:finish  :   4:         0:  .00000000000000000000000000000000D+00:
```

run the program, start bug with the map, and you will have

```
BUG: 0: 1:start   !   1:         0:  .00000000000000000000000000000000D+00:
MAP: 0: 2:.....................................................:
COM: 0:MESSAGE      456:
BUG: 0: 1:tutorial!   2:       511:  .19990000000000000000000000000000D+04:
message can be smuggled in
BUG: 0: 2:peekaboo:   3:         7:  .00000000000000000000000000000000D+00:
BUG: 0: 2:peekaboo:   4:         8:  .00000000000000000000000000000000D+00:
COM: 0:QUIT
BUG: 0: 1:hello   :   5:         0:  .00000000000000000000000000000000D+00:
```

- Higher level BUG call with higher level 'Lid' value is activated only after receiving an appropriate 'LEVEL' command.
- BUG is active only when the arguments 'J', 'J1' and 'J2' satisfies the relation J1 $\leq$ J $\leq$ J2.

13. Edit 'bug.map' and change the line

```
BUG: 0: 1:tutorial:   2:       511:  .19990000000000000000000000000000D+04:
```

to

```
BUG: 0: 1:tutorial:   2:       511:  .20000000000000000000000000000000D+04:
```

run the program, start bug with the map, and you will have

```
BUG: 0: 1:start   !   1:         0:  .00000000000000000000000000000000D+00:
MAP: 0: 2:.....................................................................
COM: 0:MESSAGE      456:
BUG: 0: 1:tutorial!   2:       511:  .19990000000000000000000000000000D+04:
mismatch!!!!!!!!!!!!!!!!!!!!!!!!!!!xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
OLD: 0: 1:tutorial:   2:       511:  .20000000000000000000000000000000D+04:
DEB: 0: ratio = (new_rid-old_rid)/old_rid :  .500D-03
DEB: 0:                          letitgo :       -10
DEB: 0:QUIT on the mismatch
```

- If a 'bug.map' record does not match, bug can stop the execution of the program.
- With this function, BUG can help program modifier, who is trying to get an efficient procedure not changing the results.

14. Edit 'bug.map' and change the line

```
COM: 0:LETITGO      -10:
```

to make the 'bug.map'

```
COM: 0:LOOK
COM: 0:LETITGO       -3:
MEM: 0: bug.map command list
MEM: 0:  LOOK     LETITGO  MESSAGE  LEVEL    BACK     QUIT
MEM: 0:  SKIP     DUMMY
BUG: 0: 1:start   :   1:         0:  .00000000000000000000000000000000D+00:
COM: 0:LEVEL        2
COM: 0:MESSAGE      456
BUG: 0: 1:tutorial:   2:       511:  .20000000000000000000000000000000D+04:
COM: 0:QUIT
BUG: 0: 1:hello   :   3:         0:  .00000000000000000000000000000000D+00:
BUG: 0: 1:finish  :   4:         0:  .00000000000000000000000000000000D+00:
```

run the program, start bug with the map, and you will have

```
COM: 0:LOOK
COM: 0:LETITGO       -3:
BUG: 0: 1:start   !   1:         0:  .00000000000000000000000000000000D+00:
MAP: 0: 2:.....................................................................
COM: 0:MESSAGE      456:
BUG: 0: 1:tutorial!   2:       511:  .19990000000000000000000000000000D+04:
mismatch!!!!!!!!!!!!!!!!!!!!!!!!!!!xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
OLD: 0: 1:tutorial:   2:       511:  .20000000000000000000000000000000D+04:
DEB: 0: ratio = (new_rid-old_rid)/old_rid :  .500D-03
DEB: 0:                          letitgo :       -3
message can be smuggled in
BUG: 0: 2:peekaboo:   3:         7:  .00000000000000000000000000000000D+00:
BUG: 0: 2:peekaboo:   4:         8:  .00000000000000000000000000000000D+00:
COM: 0:QUIT
BUG: 0: 1:hello   :   5:         0:  .00000000000000000000000000000000D+00:
```

- Even if 'bug.map' records do not match, bug can allow the continued execution of the program by relaxing the 'LETITGO' value.
- This function is useful when you are transplanting your program on to other kind machine.

3 Formal Manual

Debugging with DEBB consists of the repetition of the following steps:

1. Preparation of a BUGged program and a BUG map file.
2. Addition of BUG commands to the BUG map.
3. Execution of the BUGged program with the BUG map.

These operations are detailed using the words listed here.

[Words]

BUG The upper case 'BUG' denotes the subroutine bug, whereas the lower case 'bug' denotes a defect of a program which causes troubles.

BUG status BUG's status is specified by four variables MODE, LEVEL, MESSAGE and LETITGO.

BUG points Points in a program where BUG is called. 'Level' is assinged to each BUG point. BUG points can be active or innactive.

BUGged program A program with BUG points.

BUG map A file refered by BUGged program. The file should be named 'bug.map'. The file should contain 'BUG lines' and 'BUG commands'.

BUG line Will be explained in this article. BUG lines can be active or innactive.

BUG command Will be explained in this article. BUG commands can be active or innactive.

3.1 BUG point

BUG point is defined in a program by calling the subroutine BUG. The generic form of BUG call is

```
call  bug ( Lid , J , J1 , J2 , Cid , Iid , Rid , Message )
```

0) 'Message' is the only output variable. Others are inputs. 'Message' should be an integer variable, should not be a constant.

1) The first argument 'Lid' should be an integer constant greater than or equal to 0. The level of a BUG point is defined by 'Lid'.

2) 'J', 'J1' and 'J2' should be integer constants or variables.

3) The fifth argument 'Cid' should be a character constant or variable of the length upto 8 byte. BUG line written at an active BUG point is marked by 'Cid'.

4) 'Iid' and 'Rid' are integer and real constant(or variable), respectively.

3.2 BUG function

The operation of BUG is dependent on its status. The status is defined by values of the following variables.

variable	range	initial/default value
MODE	'new', 'map'	
LEVEL	− 1, 0, 1, 2, ...	0
MESSAGE	any value	1
LETITGO	− 10,− 9, ..., 1, 2, 3	− 10

A BUG point is active if and only if the following two conditions are met.

- The value of status variable LEVEL is equal to or higher than the level of the BUG point.
- J1 $\leq$ J $\leq$ J2 holds.

Functin of a BUG point

1. The present value of the BUG status variable MESSAGE is returned through the 'message' argument. Write, for example, as follows

```
call bug ( 1, 0, 0, 0, 'Namae', 1496, 0.5d0, message)
  if (message.eq.1) then
    output for debugging
  end if
```

then the 'output for debugging' can be controlled by the value of the MESSAGE variable.

[sbuggle routine] There is another way of getting MESSAGE value. Call subroutine 'sbuggle' as follows:

```
call sbuggle(message)
```

This subroutine has no output. It returns 'message'. The 'message' value is set equal to the present MESSAGE value of BUG. It can be used, for example, as follows:

```
call sbuggle(message)
if (message .ge. 1) then
   print out something
end if
```

2. If a BUG point

   ```
   call  bug ( Lid , J , J1 , J2 , Cid , Iid , Rid , Message )
   ```

   is active, it prints out a BUG line

   ```
   BUG: 0:Lid:Cid:N:Iid:Rid
   ```

3. If MODE is 'map':

   - BUG reads BUG comands in the BUG map and change its status accordingly , and
   - check the next BUG line

     ```
     BUG: 0:lid:Cid:n:iid:rid
     ```

   - If Cid $\neq$ cid or Iid $\neq$ iid or Rid $\neq$ rid, it prints out a warning message.

     BUG stops the execution of the program when either of conditions is met:

     $$\begin{array}{lll} \text{LETITGO} \leq 2 & \text{and} & \text{cid} \neq \text{Cid} \\ \text{LETITGO} \leq 1 & \text{and} & \text{iid} \neq \text{Iid} \\ \text{LETITGO} \leq 0 & \text{and} & \text{r} \geq \text{LETITGO} \end{array}$$

     where

     $$r = \begin{cases} -\infty & \text{if } Rid = rid \\ 0.0 & \text{if } Rid \neq rid = 0.0 \\ \log_{10} \left| \dfrac{Rid - rid}{rid} \right| & \text{otherwise} \end{cases}$$

   - BUG never stops the program if LETITGO ≥ 3, checks only Cid if LETITGO $= 2$, Check Cid and Iid if LETITGO $= 1$.
   - Rid is checked when LETITGO ≤ 0, where the tolerance of the check is controled by the setting of LETITGO.

3.3 How to Control BUG Status

MODE: At the first level 0 BUG point, BUG asks the following question:

```
Bug ? (<Y>es/with <M>ap/<N>o bug)'
```

Answer 'y' to choose 'new', 'm' to choose 'map' MODE, respectively. Choose 'no-bug' MODE by answering 'n' if you need not 'bug'. In this case MESSAGE is fixed at 0.

LEVEL: At the first level 0 BUG point, BUG asks the following question:

```
Bug ? (<Y>es/with <M>ap/<N>o bug)'
```

- Answer 'y' then LEVEL is set equal to 1.
- Answer 'm' make BUG look for LOOK command in the BUG map. If there is a corresponding LOOK command, LEVEL is set equal to 1.
- Answer 'n' to make all BUG points inactive by fixing LEVEL at -1..
- If BUG is started in 'map' MODE, LEVEL can be controlled by the BUG command 'LEVEL'.

LETITGO: If BUG is started in 'map' MODE, it is controlled by the BUG command 'LETITGO'.

MESSAGE: If BUG is started in 'map' MODE, it is controlled by the BUG command 'MESSAGE'. It is fixed at 0 'no-bug' MODE is chosen.

3.4 BUG command

There are six BUG commands. The generic form of COMMAND is

```
COM:id:CCCC    d
```

'CCCC' is the command name; 'd' is an integer constant. There are commands which take no argument like 'QUIT' command.

[**LOOK**]

```
[LOOK] COM: 0:LOOK
```

This command should placed before every BUG line or BUG command addressed to the node of the same rank.

[**LEVEL**] The command

```
COM:0:LEVEL       i
```

is used to set BUG level at i.

[**BACK**] The command

```
COM:0:BACK
```

is used to resume the BUG level of present BUG map.

[**LETITGO**] Command

```
COM:0:LETITGO         m
```

set the status variable LETITGO of BUG at m.

[**MESSAGE**] Command

```
COM:0:MESSAGE         m
```

set the status variable MESSAGE of BUG at m.

[**QUIT**] A command

```
COM:0:QUIT
```

stops the execution of the program at the next BUG point.

3.5 BUGged program

1. There should be at least one BUG point of level 0 .
2. Unit number 92 and 91 are reserved for BUG map file and log file named bug.map nad bug.log, respectively. Don't use these unit numbers for other files.
3. 'cmbug' is reserved as the name of common area used by subroutines BUG and 'sbuggle'..
4. There should be no output lines of the form starting "BUG:", "COM:", "MAP:".
5. This subroutine can be called anywhere in a program. Of course some kind of bug is interfered with our BUG routine. BUG is for the other kind of bugs.
6. Memo written in a BUG map with the format:

   ```
   MEM:x: .........
   ```

   is reproduced in the output when the program is executed in with-map mode.
7. It is possible to add BUG points of higher level than that of the present program.

3.6 Debugging on Parallel Computers

BUG is designed so that it can be used on parallel computers with Message-Passing Interface(MPI). See Appendix A.1 for example. C version is given as Appendix A.5.

1. In parallel computing environment, each BUG in different node has its own status, works independently, and makes its own print outs.
2. Outputs from parallely running nodes can be entangled (see Appendix A.2). This can be sorted by a simple tool (Appendix A.4) as shown in Appendix A.3.
3. If you are interested in the performance of the 'rank' x node, write LOOK command

   ```
   COM: x:LOOK
   ```

   in the bug map. This command should placed before every BUG line or BUG command addressed to the node of the same rank. Otherwise, BUG points are not activated in the node.
4. In 'map'mode, BUG points active in a node reads BUG lines written by itself and skips those written by BUG activated in other nodes.
5. In 'map'mode, BUG active in a node receives BUG commands if and only if the command is addressed to the node. Use COM:x: to address the command to the node of rank x. (BUG command with $x = -1$ is addressed to all nodes of a parallel machine).
6. When BUG lines in the BUG map are exhausted, the computation in 'map' mode is continued in 'new' mode. Because of this function, execution of a BUGged program with the BUG map with a single entry

   ```
   COM:-1:LOOK
   ```

   is equivalent to the execution of the program in 'new' mode.

3.7 C version

C version 'bug' and 'sbuggle' are available. Simple.c shown in Fig.7 is a C translation of Simple.f in Fig.1. Explanations about FORTRAN version 'bug' apply to C version.

```
#include <stdio.h>
#include <math.h>
#include "cbug.h"

main()
{
   double sum,a,ipow();
   int i,mod,message;

   sum = 0.0;
   bug(0,0,0,0,"start",0,0.0,&message);
   for(i=1;i<=500;i++) {
         a = ipow(1.1,i);
         sum = sum + 1.0 / (ipow(a+1.0,3) - ipow(a,3));
         mod = i/20;
         bug(1,i-mod*20,0,0,"sum",i, sum , &message );
         if(message == 2) {
            printf(" %d  %f  %f\n",i, ipow(a+1.0,3) - ipow(a,3), ipow(a,3));
 }
   }
   printf("sum = %f\n", sum);
}

double ipow( x, p)
double x;
int p;
{
    int i;
    double s;

    if(p > 0){
       s=1.0;
       for(i=1;i<=p;i++) s*=x;
       return s;
    }
    else if(p==0) return(1.0);
    else{
       s=1.0;
       for(i=1;i<=(-p);i++) s*=x;
       return 1.0/s ;
    }
}
```

Figure 7. Simple.c

4 Source Code

This software is distributed under the condition of the Open Market Licence for software:

Open Market Licence for software(version:OML-MI-95-0)

1. On the condition that 'Copyright - OML' notice and attached statements are copied as they are, all or any portion of the said software can be copied or redistributed.
2. Modification of the software is permitted, provided that places of the modification, the date and the name of the modifier are shown conspicuously.
3. It is forbidden to apply for a patent on the software which utilizes the said software.
4. The said software is distributed without any warranty. The copyright holder takes no responsibility for any damage caused by the use of the said software.
5. Any conversational application made utilizing the said software has to show 'Copyright - OML' notice and attached statements conspicuously when it is started up.

The copyright notice of 'bug.f' is given as follows.

```
c  bug and sbuggle, two FORTRAN subroutines for bug hunting
c  version 4-MPI
c      Copyright_OML 1999 M. Ishiguro
c       The source code of this subroutine package
c       can be obtained from ISMLIB(ftp://ftp.ism.ac.jp/pub/ISMLIB/)
c       of the Institute of Statistical Mathematics without any charge.
```

```
c       On the conditions that terms of the OPEN_MARKET_LICENCE for software
c       (ftp://ftp.ism.ac.jp/pub/ISMLIB/OML/OML-SW-E-1996) are observed and
c       that the copyright holder nor the Institute of Statistical Mathematics
c       take no responsibility on any result from the use, this subroutine
c       can be used or modified freely. On the condition that this note is
c       attached as it is, this subroutine can be redistributed.
c       Appropriate reference must be made at times of publishing
c       results obtained using this subroutine.
c
c Makio Ishiguro
c The Institute Statistical Mathematics
c 4-6-7 Minimi-Azabu Minato-ku Tokyo 106-8569 Japan
c e-mail:ishiguro@ism.ac.jp
```

A Appendix: MPI programing and 'BUG'

A.1 Defining and using communicator

```
       implicit real*8 (a-h,o-z)
#ifdef MPI
#ifdef QMPIFH
       include "mpif.h"
#else
       include (mpif.h)
#endif
#endif
       integer COMM_ID, COMM_PROCS, alloc
       common COMM_ID, MYWORLD, MYPROCS, MYALLOC, MYID,
     * nmy, my(0:4), alloc(0:4)

#ifdef MPI
       call mpi_init (IERR)
#endif
       call bug(0, 0, 0, 0, 'Init', 0, 0.0d0, message)

#ifdef MPI
       MYWORLD = MPI_COMM_WORLD
       MYALLOC = 0
       call mpi_comm_rank (MPI_COMM_WORLD , COMM_ID , IERR)
       call mpi_comm_size (MPI_COMM_WORLD , COMM_PROCS , IERR)
#else
       COMM_ID=0
       COMM_PROCS=1
#endif

       MYID = COMM_ID
       MYPROCS = COMM_PROCS
       nmy = 1
       my(0)=0
       alloc(0)=0

       call bug(1, 0, 0, 0, 'rank', MYID, 0.0d0, message)

c  ********* costomize here !!
       nlogical = 3
       alloc(1)=0
       alloc(2)=0

       if(2 .le. COMM_PROCS .and. COMM_PROCS .le. 3) then
          nmy = 2
          my(1) = COMM_PROCS-1
          alloc(1) = 0
          alloc(2) = 1
       end if
       if(COMM_PROCS .gt. 3) then
          nmy = 3
          my(1) = (COMM_PROCS-1)/3
          my(2) = COMM_PROCS-1
          alloc(1) = 1
          alloc(2) = 2
       end if
c  *********

       if(nmy .gt. 1) then
          my(nmy)=COMM_PROCS
          do 10 i=2,nmy
             if(my(i-1) .le. MYID .and. MYID .lt. my(i)) then
                MYALLOC = my(i-1)
             end if
   10     continue
#ifdef MPI
          call mpi_comm_split(MPI_COMM_WORLD,MYALLOC,MYID,
     *    MYWORLD,IERR)
          call mpi_comm_rank (MYWORLD , MYID , IERR)
```

```
         call mpi_comm_size (MYWORLD , MYPROCS , IERR)
#endif
       end if
       if(COMM_ID .eq. 0) then
          write(6,'(/,'' MPI setup'')')
          write(6,'(/,''   COMM_PROCS = '',I3)') COMM_PROCS
          write(6,'(/,''  Logical World     My World'')')
          do 1 i = 0, nlogical - 1
             write(6,'(7x,I2,13x,I2)') i,alloc(i)
    1     continue
          write(6,'(/,''     My World      node allocation'')')
          do 2 i = 0, nmy - 1
             write(6,'(7x,I2,10x,I3,'' -'',I3)') i,my(i),my(i+1)-1
    2     continue
          write(6,*)
       end if

       call sub

       call bug(1, 0, 0, 0, 'Finalize', 0, 1.0d0, message)
#ifdef MPI
       call mpi_finalize ( ierr )
#endif
       stop
      end

      subroutine sub
       implicit real*8 (a-h,o-z)
#ifdef MPI
#ifdef QMPIFH
       include "mpif.h"
#else
       include (mpif.h)
#endif
#endif
       integer B,E,NN,n,ircnt(0:100),ista,iend,idisp(0:100),
     * jx,i,j,ivals(100)
       integer message, COMM_ID,alloc
       common COMM_ID, MYWORLD, MYPROCS, MYALLOC, MYID,
     * nmy, my(0:4), alloc(0:4)

       message = 1
       call bug(1, 0, 0, 0, 'MYALLOC', MYALLOC, 0.0d0, message)
       do 1 i=1,20
          ivals(i)=0
    1  continue
       call sbuggle(message)
       if(message .gt. 0) then
          write(6,'('' OUT:'',I2,'':A:'',20I3)') COMM_ID,
     *    (ivals(i),i=1,20)
       end if

       do 100 logical = 0,1
          if(logical .eq. 0) then
             B = 1
             E=6
          end if
          if(logical .eq. 1) then
             B = 7
             E=20
          end if

          if(MYALLOC .eq. my(alloc(logical))) then
             NN = E - B + 1
             n = NN/MYPROCS
             do 101 i=1,NN-MYPROCS*n
                ircnt(i-1)=n+1
  101        continue
             do 102 i = NN-MYPROCS*n+1,MYPROCS
                ircnt(i-1)=n
  102        continue
             do 103 i=0,MYPROCS
                ista=B
                do 1031 j=1,i
                   ista = ista + ircnt(j-1)
 1031           continue
                idisp(i)=ista-1
  103        continue
             ista=B
             do 104 i=1,MYID
                ista = ista + ircnt(i-1)
  104        continue
             iend=ista+ircnt(MYID)-1
             call bug(1, 0, 0, 0, 'ista', ista, 0.0d0, message)
             do 105 jx=ista,iend
                ivals(jx)=jx
```

```
  105        continue
     call sbuggle(message)
             if(message .gt.  0) then
                write(6,'('' OUT:'',I2,'':B:'',20I3)') COMM_ID,
     *          (ivals(i),i=1,20)
     end if
     call bug(1, 0, 0, 0, 'GATHER', MYALLOC, 0.0d0, message)
#ifdef       MPI
             call MPI_Allgatherv(ivals(ista),ircnt(MYID),MPI_INTEGER,
     *       ivals,ircnt,idisp,MPI_INTEGER,MYWORLD,IERR)
#endif
             call sbuggle(message)
             if(message .gt.  0) then
                write(6,'('' OUT:'',I2,'':C:'',20I3)') COMM_ID,
     *          (ivals(i),i=1,20)
     end if
          end if
  100  continue

#ifdef MPI
       do 200 logical=0,1
          if(logical .eq. 0) then
             B = 1
             E = 6
          end if
          if(logical .eq. 1) then
             B = 7
             E=20
          end if
          call bug(1, 0, 0, 0, 'BCAST', logical, 0.0d0, message)
          if(nmy .gt.  1) then
             call MPI_Bcast(ivals(B),E-B+1,MPI_INTEGER,
     *       my(alloc(logical)), MPI_COMM_WORLD,IERR)
          end if
 200   continue
#endif
       call sbuggle(message)
       if(message .gt.  0 .or. COMM_ID .eq. 0) then
          write(6,'('' OUT:'',I2,'':D:'',20I3)') COMM_ID,
     *    (ivals(i),i=1,20)
       end if
       return
      end
```

Figure 8. MPI.f

A.2 Output

```
DEBugging Bug, version 4-MPI             1

Init: Bug ? (<Y>es / with <M>ap / <N>o bug )
Init nullified at DEBB.

MPI setup

   COMM_PROCS =   7

 Logical World     My World
       0              0
       1              1
       2              2

    My World      node allocation
       0            0 -  1
       1            2 -  5
       2            6 -  6

OUT: 0:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Figure 9. Output of MPI.f in 'no-bug' mode

```
DEBugging Bug, version 4-MPI             1

Init: Bug ? (<Y>es / with <M>ap / <N>o bug )
DEBB started
COM: 0:LOOK
COM: 0:LETITGO      -10:
MEM: 0:bug.map command list
MEM: 0: LOOK     LETITGO  MESSAGE  LEVEL    BACK     QUIT
MEM: 0: SKIP     DUMMY
BUG: 5: 1:Init    :   1:         0: 0.0                                    :
BUG: 0: 1:Init    :   1:         0: 0.0                                    :
BUG: 2: 1:Init    :   1:         0: 0.0                                    :
BUG: 1: 1:Init    :   1:         0: 0.0                                    :
BUG: 4: 1:Init    :   1:         0: 0.0                                    :
BUG: 6: 1:Init    :   1:         0: 0.0                                    :
BUG: 3: 1:Init    :   1:         0: 0.0                                    :
```

```
BUG: 5: 1:rank    :   2:          5: 0.0                                        :
BUG: 0: 1:rank    :   2:          0: 0.0                                        :
BUG: 2: 1:rank    :   2:          2: 0.0                                        :
BUG: 1: 1:rank    :   2:          1: 0.0                                        :
BUG: 4: 1:rank    :   2:          4: 0.0                                        :
BUG: 6: 1:rank    :   2:          6: 0.0                                        :
BUG: 3: 1:rank    :   2:          3: 0.0                                        :
BUG: 4: 1:MYALLOC :   3:          2: 0.0                                        :
BUG: 2: 1:MYALLOC :   3:          2: 0.0                                        :

BUG: 3: 1:MYALLOC :   3:          2: 0.0                                        :
BUG: 5: 1:MYALLOC :   3:          2: 0.0                                        :
BUG: 1: 1:MYALLOC :   3:          0: 0.0                                        :
BUG: 6: 1:MYALLOC :   3:          6: 0.0                                        :
OUT: 4:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
OUT: 5:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
MPI setup
OUT: 1:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
OUT: 3:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
OUT: 6:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
OUT: 2:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 4: 1:ista    :   4:         15: 0.0                                        :

BUG: 2: 1:ista    :   4:          7: 0.0                                        :
BUG: 3: 1:ista    :   4:         11: 0.0                                        :
BUG: 6: 1:BCAST   :   4:          0: 0.0                                        :
BUG: 5: 1:ista    :   4:         18: 0.0                                        :
BUG: 1: 1:ista    :   4:          4: 0.0                                        :
OUT: 3:B:  0  0  0  0  0  0  0  0  0  0 11 12 13 14  0  0  0  0  0  0
OUT: 2:B:  0  0  0  0  0  0  7  8  9 10  0  0  0  0  0  0  0  0  0  0
OUT: 4:B:  0  0  0  0  0  0  0  0  0  0  0  0  0  0 15 16 17  0  0  0
OUT: 5:B:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 18 19 20
   COMM_PROCS =   7
OUT: 1:B:  0  0  0  4  5  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 3: 1:GATHER  :   5:          2: 0.0                                        :
BUG: 4: 1:GATHER  :   5:          2: 0.0                                        :

BUG: 2: 1:GATHER  :   5:          2: 0.0                                        :
BUG: 1: 1:GATHER  :   5:          0: 0.0                                        :
BUG: 5: 1:GATHER  :   5:          2: 0.0                                        :
 Logical World     My World
OUT: 3:C:  0  0  0  0  0  0  7  8  9 10 11 12 13 14 15 16 17 18 19 20
OUT: 5:C:  0  0  0  0  0  0  7  8  9 10 11 12 13 14 15 16 17 18 19 20
      0                0
OUT: 2:C:  0  0  0  0  0  0  7  8  9 10 11 12 13 14 15 16 17 18 19 20
OUT: 4:C:  0  0  0  0  0  0  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 3: 1:BCAST   :   6:          0: 0.0                                        :
      1                1
BUG: 5: 1:BCAST   :   6:          0: 0.0                                        :
BUG: 4: 1:BCAST   :   6:          0: 0.0                                        :
BUG: 2: 1:BCAST   :   6:          0: 0.0                                        :
      2                2

    My World      node allocation
       0            0 -  1
       1            2 -  5
       2            6 -  6

BUG: 0: 1:MYALLOC :   3:          0: 0.0                                        :
OUT: 0:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 0: 1:ista    :   4:          1: 0.0                                        :
OUT: 0:B:  1  2  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 0: 1:GATHER  :   5:          0: 0.0                                        :
OUT: 0:C:  1  2  3  4  5  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
OUT: 1:C:  1  2  3  4  5  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 0: 1:BCAST   :   6:          0: 0.0                                        :
BUG: 1: 1:BCAST   :   6:          0: 0.0                                        :
BUG: 3: 1:BCAST   :   7:          1: 0.0                                        :
BUG: 2: 1:BCAST   :   7:          1: 0.0                                        :
BUG: 5: 1:BCAST   :   7:          1: 0.0                                        :
BUG: 6: 1:BCAST   :   5:          1: 0.0                                        :
BUG: 1: 1:BCAST   :   7:          1: 0.0                                        :
BUG: 0: 1:BCAST   :   7:          1: 0.0                                        :
BUG: 4: 1:BCAST   :   7:          1: 0.0                                        :
OUT: 5:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
OUT: 3:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
OUT: 0:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
OUT: 2:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
OUT: 6:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
OUT: 1:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
OUT: 4:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 5: 1:Finalize:   8:          0: 0.1000000000000000000                      D+01:
BUG: 0: 1:Finalize:   8:          0: 0.1000000000000000000                      D+01:
BUG: 2: 1:Finalize:   8:          0: 0.1000000000000000000                      D+01:
BUG: 6: 1:Finalize:   6:          0: 0.1000000000000000000                      D+01:
BUG: 1: 1:Finalize:   8:          0: 0.1000000000000000000                      D+01:
BUG: 4: 1:Finalize:   8:          0: 0.1000000000000000000                      D+01:
BUG: 3: 1:Finalize:   8:          0: 0.1000000000000000000                      D+01:
```

Figure 10. Output of MPI.f in ‘new’ mode

A.3 Sorted output

```
DEBugging Bug, version 4-MPI            1

Init: Bug ? (<Y>es / with <M>ap / <N>o bug )
DEBB started
```

```
COM: 0:LOOK
COM: 0:LETITGO      -10:
MEM: 0:bug.map command list
MEM: 0: LOOK     LETITGO  MESSAGE  LEVEL    BACK     QUIT
MEM: 0: SKIP     DUMMY
BUG: 0: 1:Init    :   1:          0: 0.0                                    :
BUG: 0: 1:rank    :   2:          0: 0.0                                    :

MPI setup

   COMM_PROCS =   7

 Logical World     My World
       0              0
       1              1
       2              2

    My World      node allocation
       0            0 -  1
       1            2 -  5
       2            6 -  6

BUG: 0: 1:MYALLOC :   3:          0: 0.0                                    :
OUT: 0:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 0: 1:ista    :   4:          1: 0.0                                    :
OUT: 0:B:  1  2  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 0: 1:GATHER  :   5:          0: 0.0                                    :
OUT: 0:C:  1  2  3  4  5  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 0: 1:BCAST   :   6:          0: 0.0                                    :
BUG: 0: 1:BCAST   :   7:          1: 0.0                                    :
OUT: 0:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 0: 1:Finalize:   8:          0: 0.100000000000000000                   D+01:
=== node1 ========================
BUG: 1: 1:Init    :   1:          0: 0.0                                    :
BUG: 1: 1:rank    :   2:          1: 0.0                                    :
BUG: 1: 1:MYALLOC :   3:          0: 0.0                                    :
OUT: 1:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 1: 1:ista    :   4:          4: 0.0                                    :
OUT: 1:B:  0  0  0  4  5  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 1: 1:GATHER  :   5:          0: 0.0                                    :
OUT: 1:C:  1  2  3  4  5  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 1: 1:BCAST   :   6:          0: 0.0                                    :
BUG: 1: 1:BCAST   :   7:          1: 0.0                                    :
OUT: 1:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 1: 1:Finalize:   8:          0: 0.100000000000000000                   D+01:
=== node2 ========================
BUG: 2: 1:Init    :   1:          0: 0.0                                    :
BUG: 2: 1:rank    :   2:          2: 0.0                                    :
BUG: 2: 1:MYALLOC :   3:          2: 0.0                                    :
OUT: 2:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 2: 1:ista    :   4:          7: 0.0                                    :
OUT: 2:B:  0  0  0  0  0  0  7  8  9 10  0  0  0  0  0  0  0  0  0  0
BUG: 2: 1:GATHER  :   5:          2: 0.0                                    :
OUT: 2:C:  0  0  0  0  0  0  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 2: 1:BCAST   :   6:          0: 0.0                                    :
BUG: 2: 1:BCAST   :   7:          1: 0.0                                    :
OUT: 2:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 2: 1:Finalize:   8:          0: 0.100000000000000000                   D+01:
=== node3 ========================
BUG: 3: 1:Init    :   1:          0: 0.0                                    :
BUG: 3: 1:rank    :   2:          3: 0.0                                    :
BUG: 3: 1:MYALLOC :   3:          2: 0.0                                    :
OUT: 3:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 3: 1:ista    :   4:         11: 0.0                                    :
OUT: 3:B:  0  0  0  0  0  0  0  0  0 11 12 13 14  0  0  0  0  0  0  0
BUG: 3: 1:GATHER  :   5:          2: 0.0                                    :
OUT: 3:C:  0  0  0  0  0  0  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 3: 1:BCAST   :   6:          0: 0.0                                    :
BUG: 3: 1:BCAST   :   7:          1: 0.0                                    :
OUT: 3:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 3: 1:Finalize:   8:          0: 0.100000000000000000                   D+01:
=== node4 ========================
BUG: 4: 1:Init    :   1:          0: 0.0                                    :
BUG: 4: 1:rank    :   2:          4: 0.0                                    :
BUG: 4: 1:MYALLOC :   3:          2: 0.0                                    :
OUT: 4:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 4: 1:ista    :   4:         15: 0.0                                    :
OUT: 4:B:  0  0  0  0  0  0  0  0  0  0  0  0  0  0 15 16 17  0  0  0
BUG: 4: 1:GATHER  :   5:          2: 0.0                                    :
OUT: 4:C:  0  0  0  0  0  0  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 4: 1:BCAST   :   6:          0: 0.0                                    :
BUG: 4: 1:BCAST   :   7:          1: 0.0                                    :
OUT: 4:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 4: 1:Finalize:   8:          0: 0.100000000000000000                   D+01:
=== node5 ========================
BUG: 5: 1:Init    :   1:          0: 0.0                                    :
BUG: 5: 1:rank    :   2:          5: 0.0                                    :
BUG: 5: 1:MYALLOC :   3:          2: 0.0                                    :
OUT: 5:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 5: 1:ista    :   4:         18: 0.0                                    :
OUT: 5:B:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 18 19 20
BUG: 5: 1:GATHER  :   5:          2: 0.0                                    :
OUT: 5:C:  0  0  0  0  0  0  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 5: 1:BCAST   :   6:          0: 0.0                                    :
BUG: 5: 1:BCAST   :   7:          1: 0.0                                    :
OUT: 5:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 5: 1:Finalize:   8:          0: 0.100000000000000000                   D+01:
=== node6 ========================
BUG: 6: 1:Init    :   1:          0: 0.0                                    :
BUG: 6: 1:rank    :   2:          6: 0.0                                    :
BUG: 6: 1:MYALLOC :   3:          6: 0.0                                    :
```

```
OUT: 6:A:  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
BUG: 6: 1:BCAST   :   4:          0: 0.0                                  :
BUG: 6: 1:BCAST   :   5:          1: 0.0                                  :
OUT: 6:D:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
BUG: 6: 1:Finalize:   6:          0: 0.10000000000000000                  D+01:
=== eof ========================
```

Figure 11. Sorted output of MPI.f in 'new' mode

A.4 bugsort

```
#!/usr/local/bin/perl

$nprocs = 4;

if( @ARGV[0] eq '') {print "usage: bugsort <file> [#nprocs]\n";exit;}
$file=@ARGV[0];

if( @ARGV[1] ne '') {$nprocs = @ARGV[1]-1;}

open(OUT,'>bug.new') || die "Cannot open bug.new $!";

open(F,$file) || die "Cannot open $file: $!";
while (<F>) {
  if(/(\s*BUG: 0:.*)\r/  || /(\s*BUG: 0:.*)/
  || /(\s*MAP: 0:.*)\r/  || /(\s*MAP: 0:.*)/
  || /(\s*COM: 0:.*)\r/  || /(\s*COM: 0:.*)/
  || /(\s*MEM: 0:.*)\r/  || /(\s*MEM: 0:.*)/
  || /(\s*OUT: 0:.*)\r/  || /(\s*OUT: 0:.*)/  ) {
     print OUT "$1\n";next;
  }
  if(/(\s*BUG):(.*):(.*)\r/ || /(\s*BUG):(.*):(.*)/
  || /(\s*MAP):(.*):(.*)\r/ || /(\s*MAP):(.*):(.*)/
  || /(\s*COM):(.*):(.*)\r/ || /(\s*COM):(.*):(.*)/
  || /(\s*MEM):(.*):(.*)\r/ || /(\s*MEM):(.*):(.*)/
  || /(\s*OUT):(.*):(.*)\r/ || /(\s*OUT):(.*):(.*)/) {
     $count[$2]=$count[$2]+1;
     $record[$count[$2]+$2*1000]="$1:$2:$3\n";
     next;
  }
  if(/(.*)\r\r/ || /(.*)\r/ ) {print OUT "$1\n";next;}
  print OUT;
}
close(F);

print OUT   "\n";
for ($i = 1; $i <= $nprocs; ++$i) {
print OUT   "=== node$i ========================\n";
   for ($j = 1; $j <= $count[$i]; ++$j) {
      print OUT "$record[$j+$i*1000]";
   }
}

print OUT   "=== eof ========================\n";
close(OUT);
print "bug.new prepared\n";
```

A.5 C version

```
#include <stdio.h>

#ifdef MPI
#include "mpi.h"
#endif
#include "cbug.h"

int COMM_ID=0;
#ifdef MPI
MPI_Comm MYWORLD;
#endif
int MYPROCS,   MYALLOC,   MYID=0;
int nmy, my[5], alloc[5];

void sub();

void main(int argc, char **argv[])
{
   int COMM_PROCS=1;
   int nlogical;
   int i,message;

#ifdef MPI
   MPI_Init(&argc,argv);
#endif
```

```
   bug(0, 0, 0, 0, "Init", 0, 0.0, &message);
#ifdef MPI
   MYWORLD = MPI_COMM_WORLD;
   MYALLOC = 0;
   MPI_Comm_rank(MPI_COMM_WORLD,&COMM_ID);
   MPI_Comm_size(MPI_COMM_WORLD,&COMM_PROCS);
#else
   COMM_ID=0;
   COMM_PROCS=1;
#endif
   MYID = COMM_ID;
   MYPROCS = COMM_PROCS;
   nmy = 1;
   my[0]=0;
   alloc[0]=0;
   bug(1, 0, 0, 0, "rank", MYID, 0.0, &message);
/*  ********* costomize here !!*/
   nlogical = 3;
   alloc[1]=0;
   alloc[2]=0;
   if(2 <= COMM_PROCS && COMM_PROCS <= 3){
      nmy = 2;
      my[1] = COMM_PROCS-1;
      alloc[1] = 0;
      alloc[2] = 1;
   }
   if(3 < COMM_PROCS){
      nmy = 3;
      my[1] = (COMM_PROCS-1)/3;
      my[2] = COMM_PROCS-1;
      alloc[1] = 1;
      alloc[2] = 2;
   }
/*  ********* */
   if(nmy > 1) {
      my[nmy]=COMM_PROCS;
      for(i=2;i<=nmy;i++) {
         if(my[i-1] <= MYID && MYID < my[i]) {
            MYALLOC = my[i-1];
         }
      }
#ifdef MPI
      MPI_Comm_split(MPI_COMM_WORLD,MYALLOC,MYID,&MYWORLD);
      MPI_Comm_rank(MYWORLD,&MYID);
      MPI_Comm_size(MYWORLD,&MYPROCS);
#endif
   }
   if(COMM_ID == 0) {
      printf("\nMPI setup\n   COMM_PROCS = %d\n", COMM_PROCS);
      printf("\n Logical World     My World\n");
      for(i=0;i<=nlogical-1;i++)
      printf("     %2d            %2d\n",i,alloc[i]);
      printf("\n My World     node allocation\n");
      for(i=0;i<=nmy-1;i++)
      printf("     %2d         %3d -%3d\n",i,my[i],my[i+1]-1);
      printf("\n");
   }
   sub();
   bug(1, 0, 0, 0, "Finalize", 0, 1.0, &message);
#ifdef MPI
   MPI_Finalize();
#endif
}

void sub()
{
   float x;
   int B,E,NN,n,ircnt[100],ista,iend,idisp[100],jx,i,j,ivals[100];
   int logical,message=1;
   bug(1, 0, 0, 0, "MYALLOC", MYALLOC, 0.0, &message);
   for(i=1;i<=20;i++) ivals[i]=0;
   sbuggle(&message);
   if(message > 0) {
```

```
      printf(" OUT:%2d:A:",COMM_ID);
      for(i=1;i<=20;i++) printf("%2d ",ivals[i]);printf("\n");
   }
   for(logical=0;logical<=1;logical++){
      if(logical == 0) {B = 1; E=6;}
      if(logical == 1) {B = 7; E=20;}
      if(MYALLOC == my[alloc[logical]]) {
         NN = E - B + 1;
         n=NN/MYPROCS;
         for(i=1;i<=NN-MYPROCS*n;i++) ircnt[i-1]=n+1;
         for(i=NN-MYPROCS*n+1;i<=MYPROCS;i++) ircnt[i-1]=n;
         for(i=0;i<MYPROCS;i++){
            ista=B;
            for(j=1;j<=i;j++) ista+=ircnt[j-1];
            idisp[i]=ista-1;
         }
         ista=B;
         for(i=1;i<=MYID;i++) ista+=ircnt[i-1];
         iend=ista+ircnt[MYID]-1;
         bug(1, 0, 0, 0, "ista", ista, 0.0, &message);
         for(jx=ista; jx<=iend; jx++){
            ivals[jx]=jx;
         }
 sbuggle(&message);
         if(message > 0) {
            printf(" OUT:%2d:B:",COMM_ID);
            for(jx=1; jx<=20; jx++) printf("%2d ",ivals[jx]);
            printf("\n");
 }
 bug(1, 0, 0, 0, "GATHER", MYALLOC, 0.0, &message);
#ifdef   MPI
         MPI_Allgatherv(ivals+ista,ircnt[MYID],MPI_INT,
                  ivals+1,ircnt,idisp,MPI_INT,MYWORLD);
#endif
         sbuggle(&message);
         if(message > 0) {
    printf(" OUT:%2d:C:",COMM_ID);
            for(jx=1; jx<=20; jx++) printf("%2d ",ivals[jx]);
            printf("\n");
 }
      }
   }
#ifdef MPI
   for(logical=0;logical<=1;logical++){
      if(logical == 0) {B = 1; E=6;}
      if(logical == 1) {B = 7; E=20;}
      bug(1, 0, 0, 0, "BCAST", logical, 0.0, &message);
      if(nmy > 1) {
         MPI_Bcast(ivals+B,E-B+1, MPI_INT, my[alloc[logical]],
         MPI_COMM_WORLD);
      }
   }
#endif
   sbuggle(&message);
   if(message > 0 || COMM_ID == 0) {
      printf(" OUT:%2d:D:",COMM_ID);
      for(jx=1; jx<=20; jx++) printf("%2d ",ivals[jx]);
      printf("\n");
   }
}
```