# FROM SERIAL FORTRAN TO MPI PARALLEL FORTRAN AT THE IAS: SOME ILLUSTRATIVE EXAMPLES

Let's begin with a simple trapezoidal integration program.
This version creates 'nbin' bins, and sums the function `fcn'
at the midpoints, times the bin width, with a single do loop.
Output is written to the screen and to the file `outdat.txt'.

program in file `trapserialone.f':

```
*****************************************************
c serial trapezoidal program - one loop for integration
        implicit real(8) (a-h,o-z)
        implicit integer (i-n)
        dimension ivold(1:2,1:100000)
        external fcn
        open(unit=10,file='outdat.txt')
        nbin=20000
        rescale=nbin
        iprocess=64
        total=0.d0
        do 10 i=1,nbin
        ivold(1,i)=i-1
        ivold(2,i)=i
   10 continue
        do 8000 jjj=1,nbin
        xmin=ivold(1,jjj)
        xmax=ivold(2,jjj)
        xmin=xmin/rescale
        xmax=xmax/rescale
        xhalf=.5d0*(xmin+xmax)
        out=(xmax-xmin)*fcn(xhalf)
        total=total+out
 7449 continue
 8000 continue
        write(10,1001) total
        print 1001,total
 1001 format(1x,'total=',d30.14)
        stop
        end


        function fcn(x)
        real(8) x,fcn
        fcn=x**3
        return
        end
*******************************************************
```

```
To compile:  ifort -o trapserialone  trapserialone.f
To run:  ./trapserial
```

If you omit the switch `-o trapserialone', the executable
version will appear in the file `a.out', and to run
you use: ./a.out

This runs interactively; `print' puts output on your screen,
while `write' writes output to the file `outdat.txt', which
you can access using `emacs'.

*********************************************************

The first step to write a parallel code is to rewrite the
serial code as a new serial code, with two do loops,
distributing the calculation over 'jprocess=iprocess-1' inner loops.
Note that `imn=nbin/jprocess+1' is the smallest integer such
that `jprocess' times `imn' is larger than `nbin'.  So the double
loop covers all the bins; to keep it from going outside the bin
range, I have included the statements

`if(jjj.gt.nbin) then/go to 7449/else/continue/end if'
This program, in file 'trapeserialtwo.f', runs in serial
mode and gives the same output as `trapserialone.f':

```
***********************************************************
c serial trapezoidal program - two loops for integration
        implicit real(8) (a-h,o-z)
        implicit integer(4) (i-n)
        dimension ivold(1:2,1:100000)
        external fcn
        open(unit=10,file='outdat.txt')
        nbin=20000
        rescale=nbin
        iprocess=64
        total=0.d0
        do 10 i=1,nbin
        ivold(1,i)=i-1
        ivold(2,i)=i
   10 continue
        jprocess=iprocess-1
        imn=nbin/jprocess+1
        do 8000 lprocess=1,jprocess
        jjjmin=1+(lprocess-1)*imn
        jjjmax=lprocess*imn
        do 8000 jjj=jjjmin,jjjmax
        if(jjj.gt.nbin) then
                go to 7449
        else
                continue
        end if
        xmin=ivold(1,jjj)
        xmax=ivold(2,jjj)
        xmin=xmin/rescale
        xmax=xmax/rescale
        xhalf=.5d0*(xmin+xmax)
        out=(xmax-xmin)*fcn(xhalf)
        total=total+out
 7449 continue
 8000 continue
        write(10,1001) total
        print 1001,total
 1001 format(1x,'total=',d30.14)
        stop
        end


        function fcn(x)
        real(8) x,fcn

        fcn=x**3
        return
        end
*************************************************************
```

Now we are ready to convert this code to MPI parallel fortran.
(NOTE: to run on the cluster you must request access from SNS Computing
by sending an e-mail to `help@sns.ias.edu'.)

The 'include' statement goes after the type declarations but before
the first executable statement. I have put the `call MPI' statements
immediately following the serial statements 'nbin=20000/rescale=nbin',
but before the first statment that refers to 'imy_rank' and/or 'iprocess'.
I have placed the finalizing `call MPI' just before `stop', but more
generally it can be followed by any serial code that does not refer to the
processor rank or number of processors.

Processor '0' is used only to total the results of the various processors,
and print output, which is why I have the lines `if(imy_rank.eq.0) go to 8021)'
and `8021 continue' I have removed the outer loop `do 8000 lprocess=1,jprocess'

```
from the two loop serial program, since MPI does this by feeding the program to
all of the processors, which now work in parallel do do the segements that were
done sequentially in the serial program.

After `8021 continue', I have put the statments that have processor '0
receive and sum the outputs from  processor '1' through processor 'jprocess',
and have the processors of nonzero rank send their outputs to processor '0'.
Finally, I have processor '0' print the output.

This program is set up for double precision.  For single precision, change
'real(8)' to 'real(4)', and change the type declaration `MPI_DOUBLE_PRECISION'
to 'MPI_REAL'. To send an integer, the type declaration is `MPI_INTEGER'.
MPI does not support quadruple precision, i.e., real(16).  If this accuracy
is needed for parts of your program, convert the numbers to real(8) before
sending or receving with MPI.   For a manual which lists the MPI fortran type declarations
on page 9,
see:  http://software.intel.com/en-us/forums/showthread.php?t=62806
Another useful source for MPI fortran is obtained from the link:
http://www.cs.usfca.edu/~peter/ppmpi/

The argument 73 of the send and receive statements is a `tag', which is not used in
this program.

Program in file `mpitrap.f':
**********************************************************************
c mpi trapezoidal program
        implicit real(8) (a-h,o-z)
        implicit integer(4) (i-n)
        dimension ivold(1:2,1:100000)
        external fcn
        include 'mpif.h'
        nbin=20000
        rescale=nbin
        call MPI_Init(ierr)
        call MPI_Comm_rank(MPI_COMM_WORLD,imy_rank,ierr)
        call MPI_Comm_size(MPI_COMM_WORLD,iprocess,ierr)
        if(imy_rank.eq.0) go to 8021

        do 10 i=1,nbin
        ivold(1,i)=i-1
        ivold(2,i)=i
   10 continue
        jprocess=iprocess-1
        imn=nbin/jprocess +1
        lprocess=imy_rank
        jjjmin=1+(lprocess-1)*imn
        jjjmax=lprocess*imn
        out=0.d0
        do 8000 jjj=jjjmin,jjjmax
        if(jjj.gt.nbin) then
                go to 7449
        else
                continue
        end if
        xmin=ivold(1,jjj)
        xmax=ivold(2,jjj)
        xmin=xmin/rescale
        xmax=xmax/rescale
        xhalf=.5d0*(xmin+xmax)
        trap=(xmax-xmin)*fcn(xhalf)
        out=out+trap
 7449 continue
 8000 continue
 8021 continue
        if(imy_rank.eq.0) then
        total=0.d0
        do 20 isource=1,iprocess-1
        call MPI_Recv(out,1,MPI_DOUBLE_PRECISION,isource,73,MPI_COMM_WORLD,
        istatus,ierr)
        total=total+out
   20 continue
        else if(imy_rank.ne.0) then
        call MPI_Send(out,1,MPI_DOUBLE_PRECISION,0,73,MPI_COMM_WORLD,ierr)
        end if
        if(imy_rank.eq.0) then
        print 1001,total
        end if
 1001 format(1x,'total=',d30.14)
        call MPI_FINALIZE(ierr)
        stop
        end


        function fcn(x)
        real(8) x,fcn
        fcn=x**3
        return
        end
*******************************************************************
```

To compile this, use:

```
mpif90 -o mpitrap  mpitrap.f
```

Then write a  shell script in the file `mpitrap.sh':
(This is the shell script for 'mpihello' in the SNS help page,

with 'mpitrap' substituted for 'mpihello'.)

```
*******************************************************************
#!/bin/bash
#$ -N mpitrap
#$ -pe orte 16
#$ -cwd
#$ -V
#$ -R y

MPI=/usr/local/openmpi/pgi/x86_64
PATH=${MPI}/bin:${PATH}
LD_LIBRARY_PATH=${MPI}/lib
mpirun ./mpitrap
********************************************************************
```

In the line `-pe orte 16',  16 is the number of processors you are
asking the cluster to use; this is where the program gets `iprocess'
This can be up to 64 with no time limits, and up to 512 with time
limits -- see the SNS help page for information on this and on the other
switches.

To run, you now use:  qsub mpitrap.sh

The `print' statement on MPI writes the output to a file, so there is
no need for a `write' statement; your output is already saved.  When
you print a hardcopy of the output, the job number appears on top, making
it easy to keep track of the output from different runs.

The rest is just as in SNS help for parallel jobs in C; you will be assigned
a job number, and can use it to access your diagnostics, job status, output from
`print', etc.  You can access the output file with either `more' or `emacs'.

```
**********************************************************************
```

Three final remarks:

(1)  If your program uses subroutines, put all MPI statements (MPI_Send, MPI_Recv, etc.)
in your main program, not in the subroutines.  You can always do this by transferring
information from the subroutines to the main program through the subroutine arguments.

(2)  If you have concerns that your program will exceed the allowed running time

(2)  If you have concerns that your program will exceed the allowed running time,
put in a timing monitor and print out a current time out at an early stage, which will
allow you to estimate the total running time.  You can look at the output file for
your job number while the job is still running.  If your estimate, based on the
curent time at a given stage of your running,  shows the program will
not finish within the cutoff time for the number of processors you requested, you can
delete the job with:  qdel ######  , with ###### your numerical job number.

For timing in fortran, you use:

```
tinit=secnds(0.0)            /at start of executable program/
tcurrent=secnds(tinit)       /in the middle of the program/
tfinal=secnds(tinit)         /at end of program, for a final running time /
```

(3)  You can debug your MPI program interactively on any of the compute servers before
submitting as a batch job to the cluster.  For the executable program `mpitrap', the
submit command for N simulated processors is:

```
mpirun -np N mpitrap
```

The integer N can take any value from 2 (the minimum for the trapezoidal program
as parallelized above) to 154 (a limit for our IAS system).

*************************************************************************

Please send me any corrections or suggested additions to this memo.

Steve Adler    3/03/2010