An Elementary Introduction to MPI Fortran Programming *

Wenqiang Feng †

Abstract

This note is a summary of my MATH 578 course in University of Tennessee at Knoxville. You can download and distribute it. Please be aware, however, that the note contains typos as well as inaccurate or incorrect description. At here, I would like to thank Dr. Vasilios Alexiades for providing his Outline of MPI parallelization [1]. I also would like to thank Jian Sun and Mustafa Elmas for the valuable disscussion and thank the generous anonymous authors for providing the detailed solutions and source code on the Internet. Without those help, this note would not have been possible to be made. In this note, I try to use the detailed demo code to show how to use each main MPI functions. If you find your work was cited in this note, please feel free to let me know.

*Key words: MPI, MPICH, FORTRAN, Finite Volume Method.

†Department of Mathematics,University of Tennessee, Knoxville, TN, 37909, wfeng@math.utk..edu

Contents

List of Figures

List of Tables

0 Preliminaries

In my opinion, the fastest way to learn MPI programming is reading demo code and writing. And it will be very helpful for beginners if you know the CPU architecture and some preliminary terminology definitions.

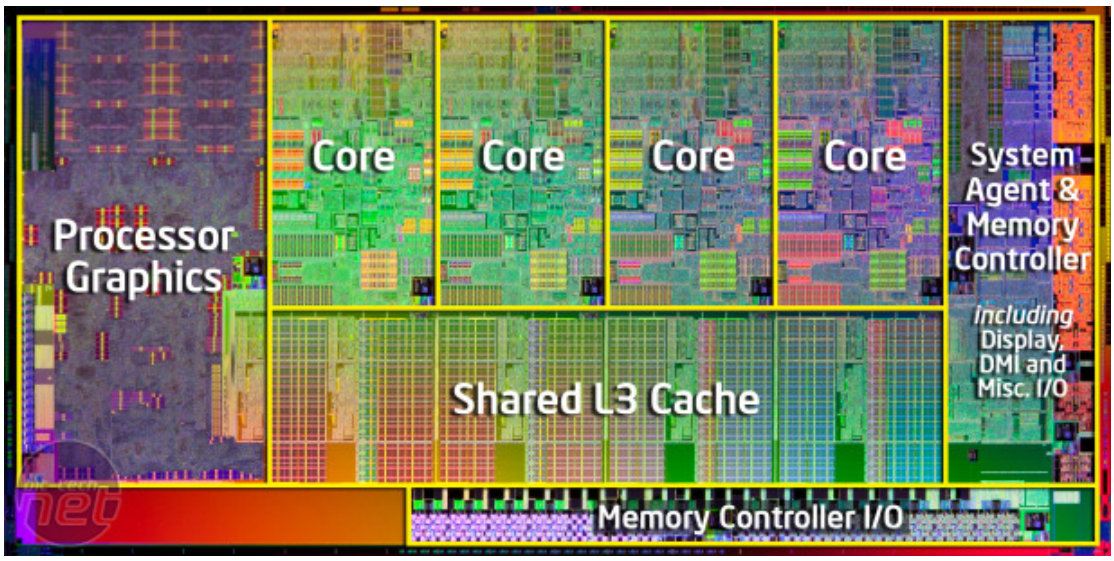


Figure 1: Intel's CPU architecture. Figure comes from [3]

The following terminology definitions come from Wikipedia or TechTerms.

Terminology definition 0.1. *Node* *A node is a basic unit used in computer science. Nodes are devices or data points on a larger network. Devices such as a personal computer, cell phone, or printer are nodes. When defining nodes on the internet, a node is anything that has an IP address. Nodes are individual parts of a larger data structure, such as linked lists and tree data structures. Nodes contain data and also may link to other nodes. Links between nodes are often implemented by pointers. [5].*

Terminology definition 0.2. *Core* *A processor core (or simply "core") is an individual processor within a CPU. Many computers today have multi-core processors, meaning the CPU contains more than one core. [2].*

Terminology definition 0.3. *(multi-core)* *A multi-core processor is a single computing component with two or more independent actual processing units (called "cores"), which are the units that read and execute program instructions.[1] The instructions are ordinary CPU instructions such as add, move data, and branch, but the multiple cores can run multiple instructions at the same time, increasing overall speed for programs amenable to parallel computing.[2] Manufacturers typically integrate the cores onto a single integrated circuit die (known as a chip multiprocessor or CMP), or onto multiple dies in a single chip package. [4].*

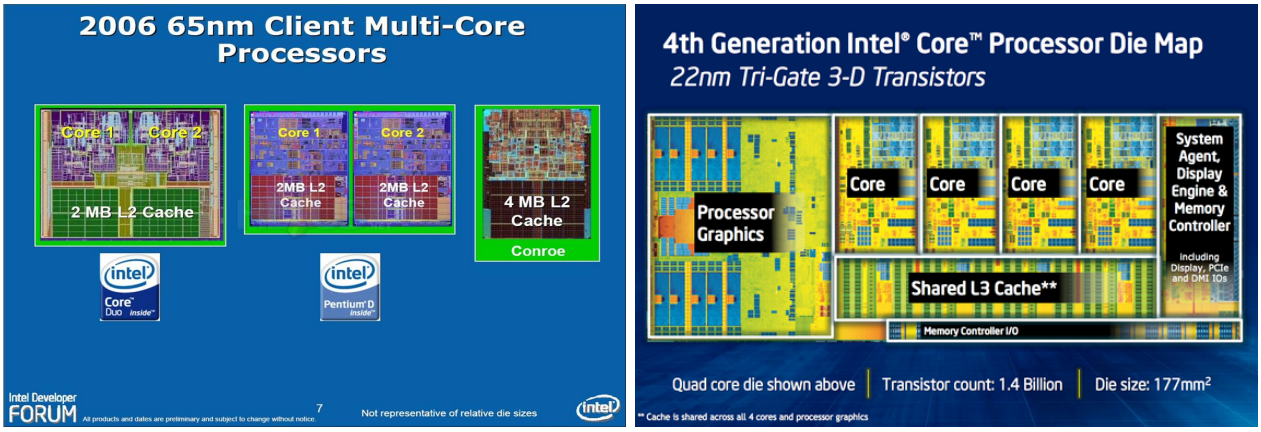


Figure 2: Intel's dual- and quad-core processors. Figures come from: [9]

Terminology definition 0.4. *(Thread) What do a t-shirt and a computer program have in common? They are both composed of many threads! While the threads in a t-shirt hold the shirt together, the threads of a computer program alllow the program to execute sequential actions or many actions at once. Each thread in a program identifies a process that runs when the program asks it to (unlike when you ask your roommate to do the dishes).*
Threads are typically given a certain priority, meaning some threads take precedence over others. Once the CPU is finished processing one thread, it can run the next thread waiting in line. However, it's not like the thread has to wait in line at the checkout counter at Target the Saturday before Christmas. Threads seldom have to wait more than a few milliseconds before they run. Computer programs that implement "multi-threading" can execute multiple threads at once. Most modern operating systems support multi-threading at the system level, meaning when one program tries to take up all your CPU resources, you can still switch to other programs and force the CPU-hogging program to share the processor a little bit. [6].

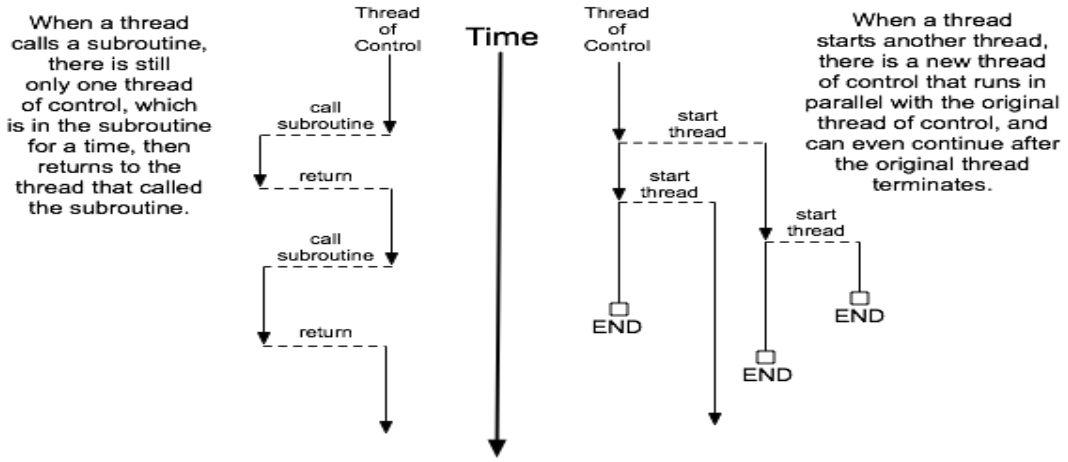


Figure 3: Threads .vs. subroutines. Figure comes from: [7]

1 MPI Introduction

Message-Passing Interface (MPI) is a message-passing library interface [11]. The goal of the MPI simply stated is to develop a widely used functions for communication between jobs that are executed on one or more processors.

In computing, SPMD (single program, multiple data) [14] technique is applied to achieve parallel execution; it is a subcategory of MIMD (multiple instruction, multiple data). In general, in order to speed up the program, the tasks are split up and run simultaneously on multiple processors.

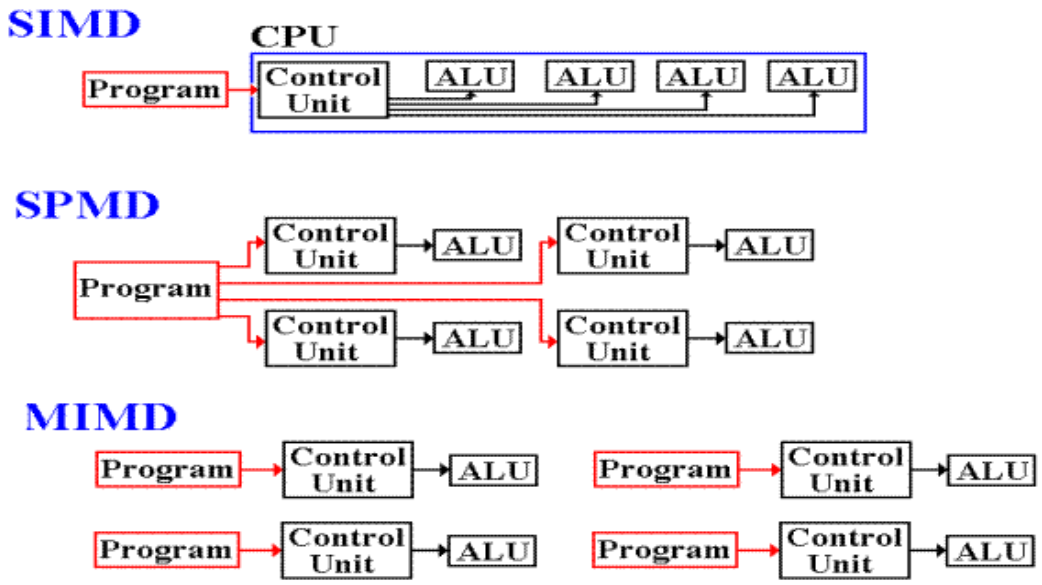


Figure 4: Main difference between the SIMD and SPMD architectures and compares each to the MIMD architecture. Figure comes from: [10]

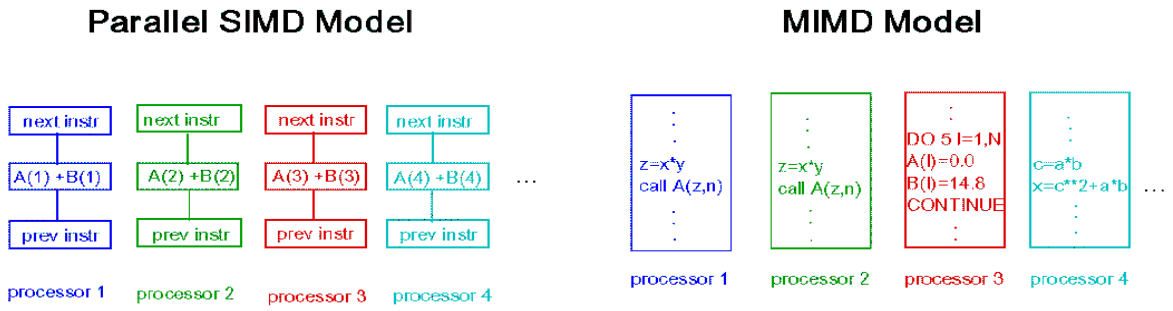


Figure 5: SIMD and MIMD. Figures come from [10]

With the SPMD technique, the program for each processor is same, only with difference inputs. That is to say the multiple instances of the same program are simultaneously executed and each instance is called an MPI process. Various communication functions are used to exchange data between MPI processes. Thanks

to A Message-Passing Interface Standard [11], the same functions from different package will do the same jobs. Three of the most popular packages are OpenMPI, MPICH and ItelMPI.

The following is a good summary for the difference between MPICH and OpenMPI from [12]:

"First, it is important to recognize how MPICH and OpenMPI are different, i.e. that they are designed to meet different needs. MPICH is supposed to be high-quality reference implementation of the latest MPI standard and the basis for derivative implementations to meet special purpose needs. OpenMPI targets the common case, both in terms of usage and network conduits.
One common complaint about MPICH is that it does not support InfiniBand, whereas OpenMPI does. However, MVAPICH and Intel MPI (among others) - both of which are MPICH derivatives - support InfiniBand, so if one is willing to define MPICH as "MPICH and its derivatives", then MPICH has extremely broad network support, including both InfiniBand and proprietary interconnects like Cray Seastar, Gemini and Aries as well as IBM Blue Gene (/L, /P and /Q). OpenMPI also supports Cray Gemini, but it is not not supported by Cray. Very recently, MPICH supports InfiniBand through a netmod, but MVAPICH2 has extensive optimizations that make it the preferred implementation in nearly all cases.
An orthogonal axis to hardware/platform support is coverage of the MPI standard. Here MPICH is far and away superior. MPICH has been the first implementation of every single release of the MPI standard, from MPI-1 to MPI-3. OpenMPI has only recently supported MPI-3 and I find that some MPI-3 features are buggy on some platforms. Furthermore, OpenMPI still does not have holistic support for MPI_THREAD_MULTIPLE, which is critical for some applications. It might be supported on some platforms but cannot generally be assumed to work. On the other hand, MPICH has had holistic support for MPI_THREAD_MULTIPLE for many years.
One area where OpenMPI used to be significantly superior was the process manager. The old MPICH launch (MPD) was brittle and hard to use. Fortunately, it has been deprecated for many years (see the MPICH FAQ entry for details). Thus, criticism of MPICH because MPD is spurius. The Hydra process manager is quite good and has the same usability and feature set as ORTE (in OpenMPI).
Here is my evaluation on a platform-by-platform basis:
Mac OS: both OpenMPI and MPICH should work just fine. If you want a release version that supports all of MPI-3 or MPI_THREAD_MULTIPLE, you probably need MPICH though. There is absolutely no reason to think about MPI performance if you're running on a Mac laptop. Linux with shared-memory: both OpenMPI and MPICH should work just fine. If you want a release version that supports all of MPI-3 or MPI_THREAD_MULTIPLE, you probably need MPICH though. I am not aware of any significant performance differences between the two implementations. Both support single-copy optimizations if the OS allows them. Linux with Mellanox InfiniBand: use OpenMPI or MVAPICH2. If you want a release version that supports all of MPI-3 or MPI_THREAD_MULTIPLE, you need MVAPICH2 though. I find that MVAPICH2 performs very well but haven't done a direct comparison with OpenMPI on InfiniBand, in part because the features for which performance matters most to me (RMA aka one-sided) have been broken in OpenMPI every time I've tried to use them. Linux with Intel/Qlogic True Scale InfiniBand: I don't have any experience with OpenMPI in this context, but MPICH-based Intel MPI is a supported product for this network and MVAPICH2 also supports it. Cray or IBM supercomputers: MPI comes installed on these machines automatically and it is based upon MPICH in both cases. Windows: I see absolutely no point in running MPI on Windows except through a Linux VM, but both Microsoft MPI and Intel MPI support Windows and are MPICH-based. In full disclosure, I currently work for Intel in a research capacity (and therefore have no special knowledge about products) and formerly worked for Argonne National Lab for five years, where I collaborated extensively with the MPICH team."

2 MPI Installation

The following steps explain how to install MPI on 64-bit Ubuntu 14.04 and 15.04 Linux. Since Intel has its own MPI package, I installed MPICH first. I have tested it on my Thinkpad W-541 with gfortran and Ifort.

2.1 OpenMPI Installation

1. Download the package: http://www.open-mpi.org/software/ompi/v1.10/
2. Unpack the downloaded file

```
tar -xvf openmpi-1.8.1.tar.gz

cd openmpi-1.8.1
```

3. Configure the installation file

```
ubuntu:   ./configure --prefix="/home/$USER/.openmpi"
Mac:    ./configure --prefix=/usr/local
```

4. Install OpenMPI (This path will take time to complete)

```
make

sudo make install
```

5. Setup path in Environment Variable
   Terminal Command:

```
vim ~/.bashrc
```

   add following lines to .bashrc

```
export PATH="$PATH:/home/$USER/.openmpi/bin"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/home/$USER/.openmpi/lib/"
```

6. Test if installation was successful

```
mpirun

--------------------------------------------------------------------------
mpirun could not find anything to do.

It is possible that you forgot to specify how many processes to run
via the "-np" argument.
--------------------------------------------------------------------------
```

7. Find link path

```
which mpirun

/home/feng/.openmpi/bin/mpirun
```

2.2 MPICH Installation

1. Download the package: https://www.mpich.org/
2. Unpack the downloaded file

```
tar -vxf mpich_3.0.4.orig.tar.gz

cd mpich-3.0.4
```

3. Configure the installation file

```
ubuntu:   ./configure
```

4. Install OpenMPI (This path will take time to complete)

```
make

sudo make install
```

5. Test if installation was successful

```
mpirun

--------------------------------------------------------------------------
mpirun could not find anything to do.

It is possible that you forgot to specify how many processes to run
via the "-np" argument.
--------------------------------------------------------------------------
```

6. Find link path

```
which mpirun

/usr/local/bin/mpirun
```

2.3 Intel MPI Installation

Intel Fortran is free for students. And it has may nice features: much more stable, much more efficient and has more debug flag than gfortran. Moreover, the installation is super easy. If you are a student, I strongly recommend you to install Intel Fortran. It will save a lot of time when you debug.

1. Download the package: https://software.intel.com/en-us/qualify-for-free-software
2. Unpack the downloaded file

```
tar -xvf parallel_studio_xe_2016_update1.tgz

cd parallel_studio_xe_2016_update1
```

3. The Installation

```
sudo ./install.sh
```

4. Setup path in Environment Variable
   Terminal Command:

   ```
   vim ~/.bashrc
   ```

   add following lines to .bashrc

   ```
   source /opt/intel/bin/compilervars.sh intel64
   export PATH=$PATH:/opt/intel/bin
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/lib/intel64
   ```

5. Test if installation was successful

   ```
   mpirun

   ----------------------------------------------------------------------------
   Usage: ./mpiexec [global opts] [exec1 local opts] : [exec2 local opts] :

   ...............................bla bla.....................................

   Intel(R) MPI Library for Linux* OS, Version 5.1.2 Build 20151015 (build id: 13147)
   Copyright (C) 2003-2015, Intel Corporation. All rights reserved.
   ----------------------------------------------------------------------------
   ```

6. Find link path

   ```
   which mpirun

   /opt/intel/compilers_and_libraries_2016.1.150/linux/mpi/intel64/bin/mpirun
   ```

3 How to Compile and Run MPI Programs in Fortran

3.1 Run a single-file code

1. Terminal Command (compile):

   ```
   mpif90 name.f90
   ```

2. Terminal Command (run):

   ```
   mpirun -np 4 ./a.out
   ```

3.2 Run a multi-file code (Makefile)

At here, I will provide two excellent templates of Makefile. The first version comes from Vasilios Alexiades [1] and the second version comes from my advisor Dr. Steven Wise.

1. Makefile (version 1):

```
#---------------------- Makefile for parallel -------------------------#
#       usage:  make compile ; make run  or  make pbs
#----------------------------------------------------------------------#

#============================== set MPI, compiler ======================#
#If your compiler is NOT on your path (for your shell) then
#   you need to insert the full path, e.g. /opt/intel/..../bin/ifort
##-----> set appropriate compiler_wrapper: mpif77 mpif90 mpicc mpic++
  COMP   = $(MPI)/bin/mpif90
##-----> set appropriate extension: f  c  cpp
  EXT    = f90
  LFLAGs =
#for C:  LFLAGs = -lm
##-------------------------- for all:
  FLAGs  = -g  $(MPIlnk)
# FLAGs  = -O3 $(MPIlnk)
  MPIlnk = -I$(MPI)/include  -L $(MPI)/lib
##---------------------> set path to openMPI on local:
  MPI  = /usr/local
#
#============================ set source code  =========================#
##--------------->set names for your PROGram and std I/O files:
  PROG = code2D
  INPUT  = ./inputdata.dat
  OUTPUT = out
##--------------------> set code components:
  CODE_o =  main.o mainMR.o mainWR.o io.o setup.o update.o messaging.o

#-==========================create executable: make compile ==========#
#------------> lines below a directive MUST start with TAB <-----------#
$(CODE_o):%.o: %.$(EXT)
 $(COMP) $(FLAGs) -c $< -o $@

compile:$(CODE_o)
# $(COMP) $(FLAGs)  $(CODE_o)  -o $(PROG).x  $(LFLAGs)
 $(COMP) $(FLAGs)  $(CODE_o)  -o $(PROG)  $(LFLAGs)
 @echo " >>> compiled on `hostname -s` with  $(COMP) <<<"
#---------------------- execute: make run -------------------------#
run:
# $(MPI)/bin/mpiexec -n  2  ./$(PROG).x < $(INPUT) > $(OUTPUT)
# $(MPI)/bin/mpirun  -np 2  ./$(PROG).x < $(INPUT) > $(OUTPUT)
 $(MPI)/bin/mpirun  -np 5  ./$(PROG) < $(INPUT)

#---------------------- execute: make pbs -------------------------#
pbs:
 @ vi PBSscript
```

```
 make clean
 qsub  PBSscript
#------------------------- clean  ----------------------------------#
clean:
 rm -f  o.*  DONE *.o watch
#-------------------------------------------------------------------#
```

2. Makefile (version 2):

```
#---------------------- Makefile for parallel -------------------------#
#       usage:  make  ;  make run  or  make pbs
#-------------------------------------------------------------------#

#============================== set MPI, compiler ====================#
#If your compiler is NOT on your path (for your shell) then
#   you need to insert the full path, e.g. /opt/intel/..../bin/ifort
##-----> set appropriate compiler_wrapper: mpif77 mpif90 mpicc mpic++

FOR = mpif90 -IMODF -JMODF

##---------------------> set path to openMPI on local:
EXE = lab9
OUTPUT = OUT/out

# OBJ has order
OBJ = OF/io.o OF/update.o OF/setup.o  OF/messaging.o OF/mainWR.o OF/mainMR.o OF/main.o

#-==========================create executable: make compile ==========#
$(EXE): $(OBJ)
 $(FOR) $(OBJ) -o $(EXE)

##-------------------- set code components:--------------------------#
OF/io.o: io.f90
 $(FOR) -c io.f90 -o OF/io.o

OF/setup.o: setup.f90
 $(FOR) -c setup.f90 -o OF/setup.o

OF/update.o: update.f90
 $(FOR) -c update.f90 -o OF/update.o

OF/messaging.o: messaging.f90
 $(FOR) -c messaging.f90 -o OF/messaging.o

OF/mainWR.o: mainWR.f90
 $(FOR) -c mainWR.f90 -o OF/mainWR.o

OF/mainMR.o: mainMR.f90
 $(FOR) -c mainMR.f90 -o OF/mainMR.o

OF/main.o: main.f90
 $(FOR) -c main.f90 -o OF/main.o
```

```
#------------> lines below a directive MUST start with TAB <-----------#
#----------------------- execute: make run -------------------------#
run:
 @mpirun -np 5 ./$(EXE) < inputdata.dat> $(OUTPUT)
#----------------------- execute: make pbs -------------------------#
pbs:
 @ vi PBSscript
 make clean
 qsub  PBSscript
#------------------------- clean  ----------------------------------#
reset:
 rm $(EXE) MODF/* OF/* ./*.mod

remove:
 rm OUT/*.dat
```

3.3 PBS script (run on sever)

The following is my PBS script for Darter which is maintained by National Institute for Computational Sciences (NICS) at the University of Tennessee. For more details, you can find in [13, 8].

```
#!/bin/bash
#PBS -A UT-TNEDU029          # account name
#PBS -l walltime=00:10:00    #  wall-clock time requested
#PBS -l size=16              #  number of processor
#PBS -N 1D_para              #  name of the job
#PBS -j oe                   # switch
#PBS -M youremail@utk.edu    # get mail notice

cd $PBS_O_WORKDIR            # change dir to job location
aprun -n 3 ./lab9 < inputdata.dat > out-log  # execution
```

- The first line in the file identifies which shell will be used for the job. In this example, bash is used but csh or other valid shells would also work.
- The second line in the file specifies the account name.
- The third line in the file states how much wall-clock time is being requested. In this example 10 minutes of wall time have been requested.
- The fourth line specifies the number of nodes and processors desired for this job. In this example, 16 processors is being requested. For some HPC may use (#PBS -l nodes=1:ppn=2) which means one node with two processors is being requested.
- The fifth line tells the cluster what's the name of the job instead of the name of the job script.
- The sixth line ombines standard output and standard error into the standard error file (eo) or the standard out file (oe).
- The seventh line tells the cluster to send the notice to your email account. You will get the notification by email when the jobs have been started or finished.
- The eighth line tells the HPC cluster to access the directory where the data is located for this job. In this example, the cluster is instructed to change the directory to the PBS_O_WORKDIR directory.

- The ninth line tells the cluster to run the program. In this example, it runs mpif90 with 3 processors on Dater, specifying lab as the argument in the current directory, with input data file inputdata.dat and output data file out-log.

4 Datatypes

4.1 Basic Datatypes

Table 1: MPI basic datatypes corresponding to Fortran datatypes

MPI datatype	Fortran datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

4.2 Derived Datatypes

1. Motivation
   - Save communication time
   - Simplify the data structure
2. Derived types
   - Vectors
   - Structs
   - Others
3. Properties
   - All derived types stored by MPI as a list of basic types and displacements (in bytes)
   - User can define new derived types in terms of both basic types and other derived types

Remark:

- for a structure, types may be different
- for an array subsection,types will be the same

MPI_TYPE_CONTIGUOUS: The simplest datatype constructor which allows replication of a datatype into contiguous locations.

Table 2: MPI_TYPE_CONTIGUOUS

MPI_TYPE_CONTIGUOUS(count, oldtype, newtype)		
IN	count	replication count (non-negative integer)
IN	oldtype	old datatype (handle)
OUT	newtype	new datatype (handle)

```
MPI_TYPE_CONTIGUOUS(2, oldtype, newtype)
 MPI_TYPE_FREE(newtype,ierr)
```

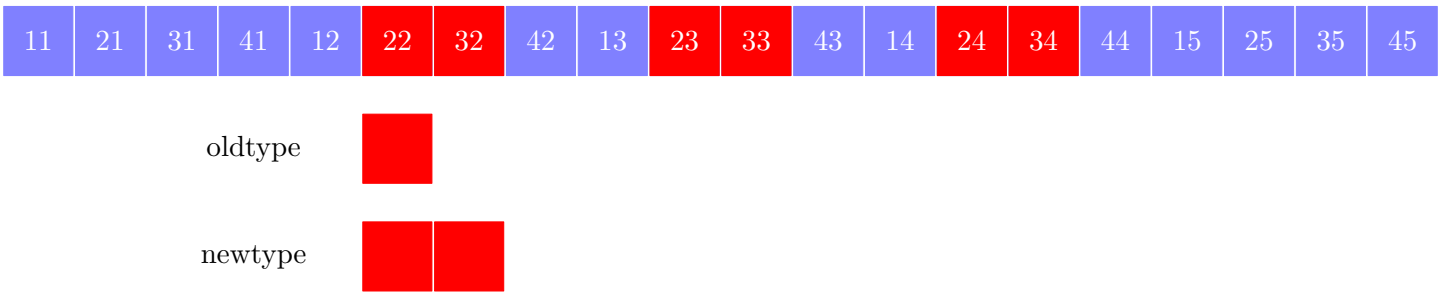


Figure 6: MPI_TYPE_CONTIGUOUS.

MPI_TYPE_VECTOR: A more general constructor that allows replication of a datatype into locations that consist of equally spaced blocks.

Table 3: MPI_TYPE_VECTOR

MPI_TYPE_VECTOR(count, blocklength, stride, oldtype, newtype)		
IN	count	number of blocks (non-negative integer)
IN	blocklength	number of elements in each block
IN	stride	number of elements between start of each block
IN	oldtype	old datatype (handle)
OUT	newtype	new datatype (handle)

```
MPI_TYPE_VECTOR(3,2,4, oldtype, newtype)
MPI_TYPE_FREE(newtype,ierr)
```

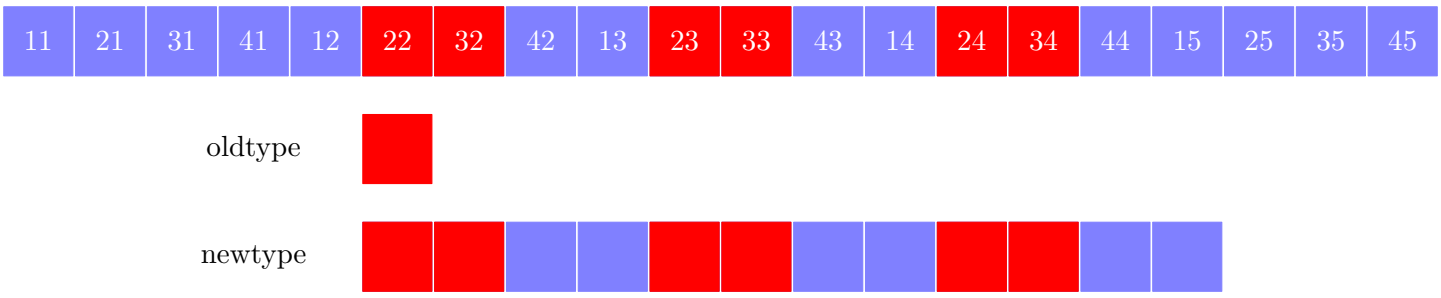


Figure 7: MPI_TYPE_VECTOR.

5 Basic Functions

Table 4: Basic computation built-in operations

Operation handle	Operation
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_PROD	Product
MPI_SUM	Sum
MPI_LAN	Logical AND
MPI_LOR	Logical OR
MPI_LXOR	Logical Exclusive OR
MPI_BAND	Bitwise AND
MPI_BOR	Bitwise OR
MPI_BXOR	Bitwise Exclusive OR
MPI_MAXLOC	Maximum value and location
MPI_MINLOC	Minimum value and location

Table 5: Basic functions for MPI programming

Operation handle	Operation
MPI_Init	Initialize MPI processes
MPI_Comm_size	Returns the number of allocated processes
MPI_Comm_rank	Returns the number of the process where the code is executed
MPI_Send	Sends a message
MPI_Recv	Receives a message
MPI_Pack	Packs a datatype into contiguous memory
MPI_Unpack	Unpack a buffer according to a datatype into contiguous memory
MPI_Bcast	Diffuses data to all processes (broadcast)
MPI_Finalize	Terminates MPI processes

5.1 MPI_COMM_INIT

MPI_COMM_INIT: Initialize the MPI execution environment

Table 6: MPI_INIT

MPI_INIT(IERR)		
OUT	ierr	error flag

Remark 5.1. *All MPI routines in Fortran (except for MPI_WTIME and MPI_WTICK) have an additional argument ierr at the end of the argument list. ierr is an integer and has the same meaning as the return value of the routine in C. In Fortran, MPI routines are subroutines, and are invoked with the call statement.*

5.2 MPI_COMM_SIZE

MPI_COMM_SIZE: Determines the size of the group associated with a communicator

Table 7: MPI_COMM_SIZE

MPI_COMM_SIZE(MPI_COMM_WORLD, nPROC, ierr)		
IN	MPI_COMM_WORLD	communicator (handle)
OUT	nPROC	number of PROCesses (integer)
OUT	ierr	error flag

5.3 MPI_COMM_RANK

MPI_COMM_RANK: Determines the rank of the calling process in the communicator

Table 8: MPI_COMM_RANK

MPI_COMM_RANK(MPI_COMM_WORLD, MyID, ierr)		
IN	MPI_COMM_WORLD	communicator (handle)
OUT	MyID	rank of the calling process in the group of comm (integer)
OUT	ierr	error flag

5.4 MPI_FINALIZE

MPI_FINALIZE: Terminates MPI execution environment

Table 9: MPI_FINALIZE

MPI_FINALIZE(ierr)		
OUT	ierr	error flag

5.5 MPI_Pack and MPI_Unpack

MPI_Pack: Packs the message in the send buffer specified by inbuf, incount, datatype into the buffer 40 space specified by outbuf and outsize.

Table 10: MPI_Pack

MPI_Pack(inbuf, incount, datatype, outbuf, outsize, position, comm)		
IN	inbuf	input buffer start (choice)
IN	incount	number of input data items
IN	datatype	data type of buffer (handle)
OUT	outbuf	output buffer start
IN	outsize	output buffer size, in bytes
INOUT	position	current position in buffer, in bytes
IN	comm	communicator for packed message (handle)

MPI_Unpack: Unpacks a message into the receive buffer specified by outbuf, outcount, datatype from the buffer space specified by inbuf and insize.

Table 11: MPI_Unpack

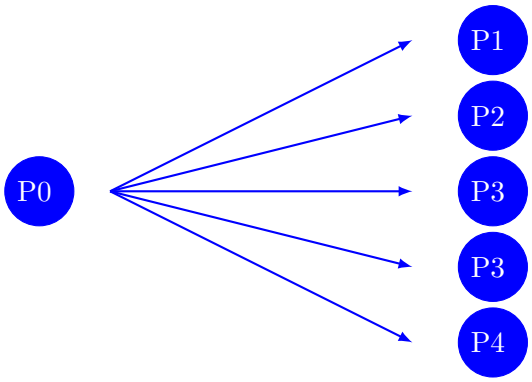
MPI_Unpack(inbuf, insize, position, outbuf, outcount, datatype, comm)		
IN	inbuf	input buffer start (choice)
IN	insize	size of input buffer, in bytes
INOUT	position	current position in buffer, in bytes
OUT	outbuf	output buffer start
IN	outcount	number of items to be unpacked
IN	datatype	datatype of each output data item
IN	comm	communicator for packed message (handle)

5.6 MPI_Bcast

MPI_Bcast: broadcasts a message from the process with rank root to all processes of the group, itself included.

Table 12: MPI_Bcast

MPI_Bcast(buffer, count, datatype, root, comm)		
INOUT	buffer	starting address of buffer (choice)
IN	count	number of entries in buffer
IN	datatype	data type of buffer (handle)
IN	root	rank of broadcast root (integer)
IN	comm	communicator (handle)



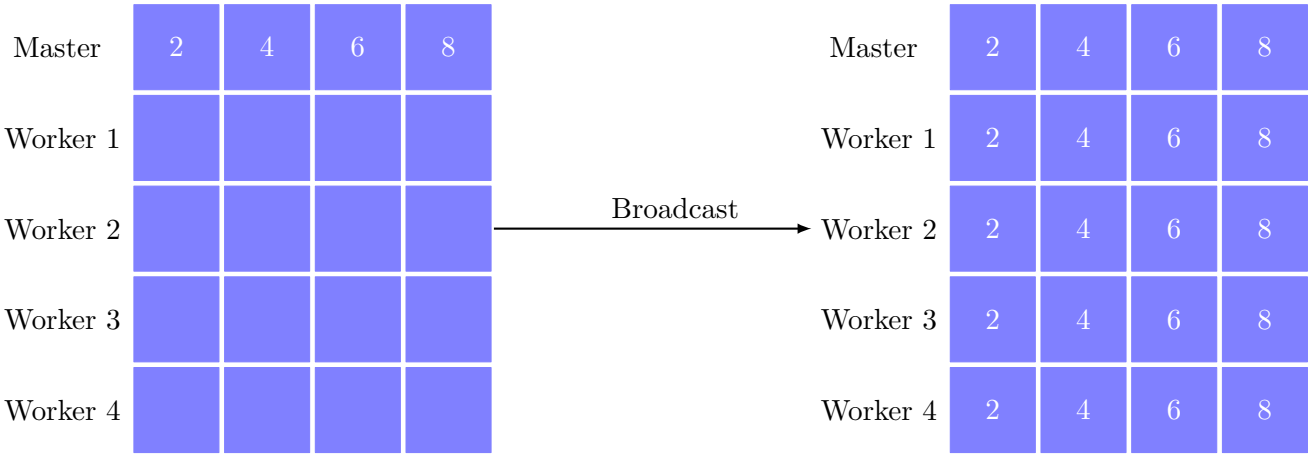


Figure 8: MPI_Broadcast.

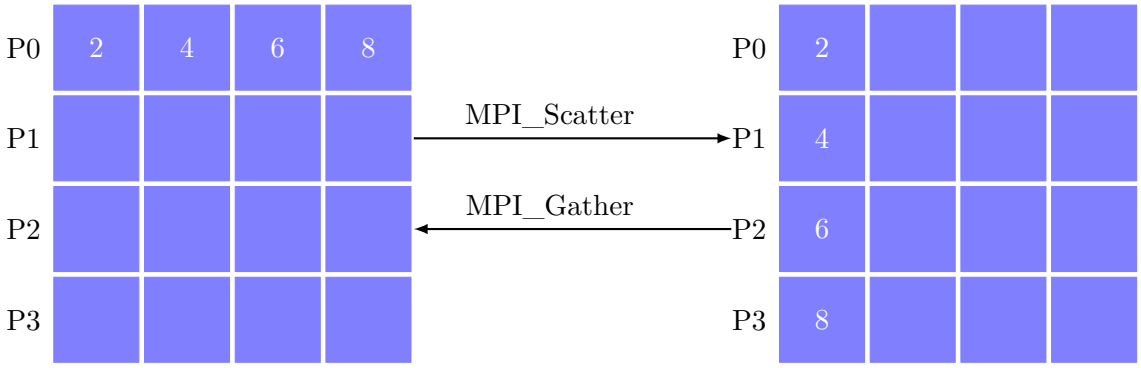


Figure 9: MPI_Scatter and MPI_Gather.

5.7 MPI_Scatter and MPI_Gather

5.8 MPI_Allgather

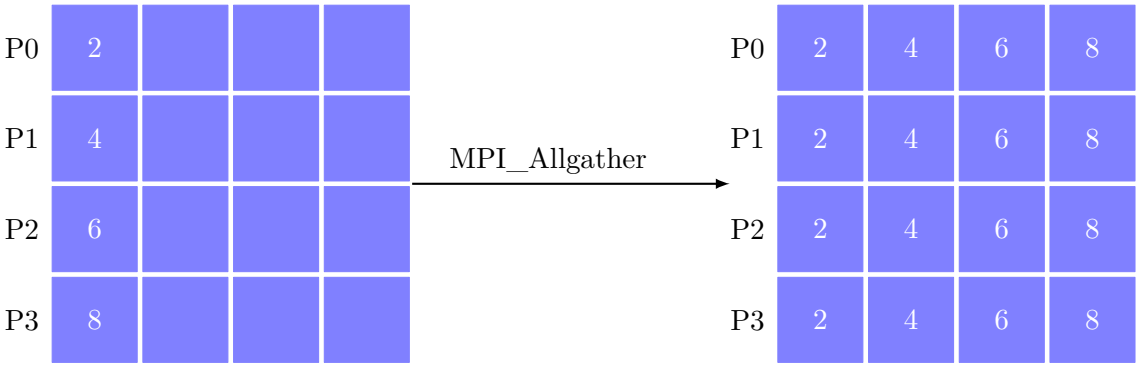


Figure 10: MPI_Allgather.

5.9 MPI_Alltoall

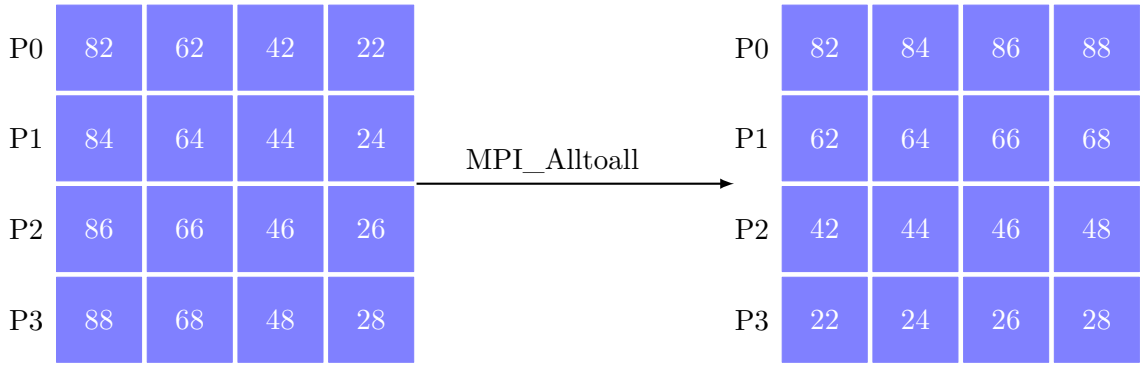


Figure 11: MPI_Alltoall.

5.10 MPI_Reduce

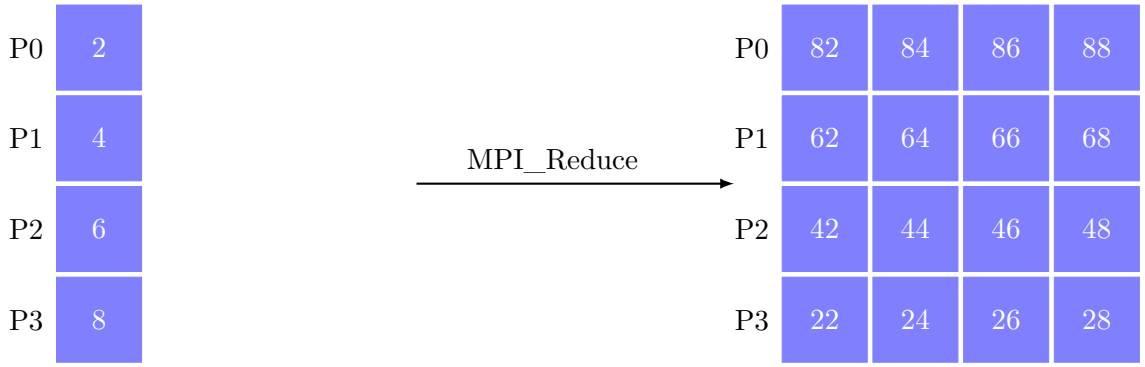


Figure 12: MPI_Reduce.

6 Hello World Demo

6.1 Makefile

```
######################################################################
#               Makefile for demo hello world
######################################################################
FOR =$(MPI)mpif90 -IMODF -JMODF
#FOR = gfortran -IMODF -JMODF -fbounds-check

# set name of the executable file
EXE = demo
```

```
##---------------------> set path to openMPI on local:
# gfortran  on my laptop
MPI  = /usr/local/bin/
# ifort on my laptop (default one)
#MPI=/opt/intel/compilers_and_libraries_2016.1.150/linux/mpi/intel64/bin/
OUTPUT = OUT/out

# set up the object
OBJ =  OF/io.o OF/mainWR.o OF/mainMR.o OF/main.o

#-==========================create executable: make  ===========#
$(EXE): $(OBJ)
$(FOR) $(OBJ) -o $(EXE)

##-------------------- set code components:---------------------#
OF/io.o: io.f90
$(FOR) -c io.f90 -o OF/io.o

OF/mainWR.o: mainWR.f90
$(FOR) -c mainWR.f90 -o OF/mainWR.o

OF/mainMR.o: mainMR.f90
$(FOR) -c mainMR.f90 -o OF/mainMR.o

OF/main.o: main.f90
$(FOR) -c main.f90 -o OF/main.o

#------------> lines below a directive MUST start with TAB <-----------#
#----------------------- execute: make run --------------------------#
run:
# @mpirun -np 5 ./$(EXE) < inputdata.dat> $(OUTPUT)
# @mpirun -np 5 ./$(EXE) < inputdata.dat
@mpirun -np 5 ./$(EXE)
reset:
rm $(EXE) MODF/* OF/* ./*.mod

remove:
rm OUT/*.dat
```

6.2 main.f90

```
!-----------------------------------------------------------
! This demo shows how to use start and end MPI
!
! Author: Wenqiang Feng
!         Department of Mathematics
!         The University of Tennessee
!
! Date  : Dec.8, 2015
!-----------------------------------------------------------
```

```
PROGRAM main
USE mainwr
USE mainmr
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
INTEGER :: ierr, nPROC,nWRs, mster, myID
REAL(r8) :: tt0,tt1
!-----------------------------------------------------------------
!       Explanation of variables for MPI  (all integers)
!-----------------------------------------------------------------
!   nPROC    = number of PROCesses = nWRs+1 to use in this run
!   nWRs     = number of workers   = nPROC-1
!   mster    = master rank (=0)
!   myID     = rank of a process (=0,1,2,...,nWRs)
!   Me       = worker's number (rank)  (=1,2,...,nWRs)
!   NodeUP   = rank of neighbor UP from Me
!   NodeDN   = rank of neighbor DN from Me
!   ierr     = MPI error flag
!-----------------------------------------------------------------

CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE( MPI_COMM_WORLD,NPROC,IERR )
mster = 0
nWRs  = nPROC - 1

Call MPI_COMM_RANK(MPI_COMM_WORLD, MYID, IERR)    !..assigns myID
!
IF( myID .EQ. mster ) THEN
!
tt0 = MPI_WTIME()
CALL MASTER( nWRs )
!
tt1 = MPI_WTIME()
       PRINT*,'>>main>> MR timing= ',tt1-tt0,' sec on ',nWRs,' WRs'
ELSE
       CALL WORKER( nWRs, myID )    !... now MPI is running ...
!
ENDIF
!
CALL MPI_FINALIZE(IERR)
!
END PROGRAM main
```

6.3 mainMR.f90

```
MODULE mainMR
USE io
CONTAINS
!
```

```
   SUBROUTINE MASTER(NWRS)
   INCLUDE 'mpif.h'
   CALL dateStampPrint
   END SUBROUTINE MASTER
!
END MODULE mainMR
```

6.4 mainWR.f90

```
MODULE mainWR
!
CONTAINS
   SUBROUTINE WORKER(nWRs, myID)
   INCLUDE 'mpif.h'
!
   INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
!
   Print *, 'Hello from worker ', myID
   END SUBROUTINE WORKER
END MODULE mainWR
```

6.5 io.f90

```
 MODULE io

CONTAINS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine  dateStampPrint
    integer    :: out_unit
    character(8)  :: date
    character(10) :: time
    character(5)  :: zone
    integer,dimension(8) :: values
    character( len = 9 ), parameter, dimension(12) :: month = (/ &
    'January  ', 'February ', 'March    ', 'April    ', &
    'May      ', 'June     ', 'July     ', 'August   ', &
    'September', 'October  ', 'November ', 'December ' /)
!
    ! call the interior function
    call date_and_time(date,time,zone,values)
    write(*,*) "################################################################"
    write(*,*) "  Demo code: MPI hello word created by Wenqiang Feng"
    write (*, '(15x,a,a1,i2,a1,i4,2x,i2,a1,i2.2,a1,i2.2,a1)' ) &
       trim ( month( values(2))),'-',values(3),'-',values(1),values(5),&
          ':', values(6), ':', values(7), '.'
    write(*,*) "Copyright (c) 2014 WENQIANG FENG. All rights reserved."
    write(*,*) "################################################################"
end subroutine  dateStampPrint
```

```
END MODULE io
```

6.6 Result

When you run the demo code with 5 processors (4 workers), then you may get the following result:

```
 Hello from worker            1
 Hello from worker            2
 Hello from worker            3
 Hello from worker            4
 ################################################################
   Demo code: MPI hello word created by Wenqiang Feng
               December-12-2015  21:35:06.
 Copyright (c) 2014 WENQIANG FENG. All rights reserved.
 ################################################################
 >>main>> MR timing=    1.6498565673828125E-004  sec on            4  WRs
```

7 MPI_PACK and MPI_UNPACK demo

7.1 Makefile

```
#########################################################################
#               Makefile for demo MPI_PACK and MPI_UNPACK
#########################################################################
FOR =$(MPI)mpif90 -IMODF -JMODF

# set name of the execution file
EXE = demo

##---------------------> set path to openMPI on local:
# gfortran  on my laptop
#MPI  = /usr/local/bin/
# ifort on my laptop (default one)
#MPI=/opt/intel/compilers_and_libraries_2016.1.150/linux/mpi/intel64/bin/
OUTPUT = OUT/out

# set up the object
OBJ =  OF/io.o OF/mainWR.o OF/mainMR.o OF/main.o

#-==========================create executable: make  ===========#
$(EXE): $(OBJ)
$(FOR) $(OBJ) -o $(EXE)

##-------------------- set code components:---------------------#
OF/io.o: io.f90
$(FOR) -c io.f90 -o OF/io.o

OF/mainWR.o: mainWR.f90
$(FOR) -c mainWR.f90 -o OF/mainWR.o
```

```
OF/mainMR.o: mainMR.f90
$(FOR) -c mainMR.f90 -o OF/mainMR.o

OF/main.o: main.f90
$(FOR) -c main.f90 -o OF/main.o

#------------> lines below a directive MUST start with TAB <-----------#
#----------------------- execute: make run --------------------------#
run:
# @mpirun -np 5 ./$(EXE) < inputdata.dat> $(OUTPUT)
# @mpirun -np 5 ./$(EXE) < inputdata.dat
@mpirun -np 5 ./$(EXE) > $(OUTPUT)
reset:
rm $(EXE) MODF/* OF/* ./*.mod

remove:
rm OUT/*.dat
```

7.2 main.f90

```
!-----------------------------------------------------------
! This demo shows how to use MPI_PACK and MPI_UNPACK
!
! Author: Wenqiang Feng
!         Department of Mathematics
!         The University of Tennessee
!
! Date  : Dec.8, 2015
!-----------------------------------------------------------
PROGRAM main
USE mainwr
USE mainmr
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
INTEGER :: ierr, nPROC,nWRs, mster, myID
REAL(r8) :: tt0,tt1
!-------------------------------------------------------------
!       Explanation of variables for MPI  (all integers)
!-------------------------------------------------------------
!   nPROC    = number of PROCesses = nWRs+1 to use in this run
!   nWRs     = number of workers   = nPROC-1
!   mster    = master rank (=0)
!   myID     = rank of a process (=0,1,2,...,nWRs)
!   Me       = worker's number (rank)  (=1,2,...,nWRs)
!   NodeUP   = rank of neighbor UP from Me
!   NodeDN   = rank of neighbor DN from Me
!   ierr     = MPI error flag
!-------------------------------------------------------------
```

```
CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE( MPI_COMM_WORLD,NPROC,IERR )
mster = 0
nWRs  = nPROC - 1

Call MPI_COMM_RANK(MPI_COMM_WORLD, MYID, IERR)    !..assigns myID
!
IF( myID .EQ. mster ) THEN
!
tt0 = MPI_WTIME()
CALL MASTER( nWRs )
!
tt1 = MPI_WTIME()
       PRINT*,'>>main>> MR timing= ',tt1-tt0,' sec on ',nWRs,' WRs'
ELSE
       CALL WORKER( nWRs, myID )    !... now MPI is running ...
!
ENDIF
!
CALL MPI_FINALIZE(IERR)
!
END PROGRAM main
```

7.3 mainMR.f90

```
MODULE mainMR
USE io
CONTAINS

SUBROUTINE MASTER(nWRS)
INCLUDE 'mpif.h'
! global parameter
INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n
REAL(r8)  :: a, b

    ! vectors for integer parameters and real parameters, respectively.
INTEGER, ALLOCATABLE, DIMENSION(:)  :: iparms
REAL(r8), ALLOCATABLE, DIMENSION(:) :: parms

! parameters for parallel
INTEGER   :: position
INTEGER, DIMENSION(100)  :: buffer

    Niparms = 1
    Nparms  = 2
    !
    n = 4         ! # of element
```

```
    a = 1.0_r8    ! left boundary
    b = 4.0_r8   ! right boundary

    Print *, 'I am master, I am sending the following data. '
Print *,  ' a=',a,' b=',b,' n=',n
! grouping
!ALLOCATE(iparms(Niparms), parms(Nparms))
!parms(1:Nparms) = (/a,b/)

! packing
position = 0
    call MPI_PACK(a,1,MPI_DOUBLE_PRECISION,buffer,100,position,MPI_COMM_WORLD,ierr)
    call MPI_PACK(b,1,MPI_DOUBLE_PRECISION,buffer,100,position,MPI_COMM_WORLD,ierr)
    call MPI_PACK(n,1,MPI_INTEGER,buffer,100,position,MPI_COMM_WORLD,ierr)

    ! communication
    call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)

!DEALLOCATE(iparms,parms)


CALL dateStampPrint
END SUBROUTINE MASTER

END MODULE mainMR
```

7.4 mainWR.f90

```
MODULE mainWR

CONTAINS
SUBROUTINE WORKER(nWRs, myID)
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n
REAL(r8)  :: a, b

! vectors for integer parameters and real parameters, respectively.
INTEGER, ALLOCATABLE, DIMENSION(:)  :: iparms
REAL(r8), ALLOCATABLE, DIMENSION(:) :: parms

! parameters for parallel
INTEGER   :: position
INTEGER, DIMENSION(100)  :: buffer
!character(len=100) :: buffer
```

```
    Niparms = 1
    Nparms  = 2

! communication
call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)
position = 0

! unpacking
    !call MPI_UNPACK(buffer,100,position,parms,Nparms,MPI_REAL,MPI_COMM_WORLD,ierr)
    !call MPI_UNPACK(buffer,100,position,n,Niparms,MPI_INTEGER,MPI_COMM_WORLD,ierr)
call MPI_UNPACK(buffer,100,position,a,1,MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierr)
    call MPI_UNPACK(buffer,100,position,b,1,MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierr)
    call MPI_UNPACK(buffer,100,position,n,1,MPI_INTEGER,MPI_COMM_WORLD,ierr)

Print *, 'I am worker ', myID, ' I received the following data from Master'
Print *,  ' a=',a,' b=',b,' n=',n
END SUBROUTINE WORKER
END MODULE mainWR
```

7.5 io.f90

```
MODULE io

CONTAINS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine  dateStampPrint
    integer    :: out_unit
    character(8)  :: date
    character(10) :: time
    character(5)  :: zone
    integer,dimension(8) :: values
    character( len = 9 ), parameter, dimension(12) :: month = (/ &
    'January  ', 'February ', 'March    ', 'April    ', &
    'May      ', 'June     ', 'July     ', 'August   ', &
    'September', 'October  ', 'November ', 'December ' /)

    ! call the interior function
    call date_and_time(date,time,zone,values)
    write(*,*) "################################################################"
    write(*,*) "         Demo code: MPI_PACK created by Wenqiang Feng"
    write (*, '(15x,a,a1,i2,a1,i4,2x,i2,a1,i2.2,a1,i2.2,a1)' ) &
       trim ( month( values(2))),'-',values(3),'-',values(1),values(5),&
          ':', values(6), ':', values(7), '.'
    write(*,*) "Copyright (c) 2015 WENQIANG FENG. All rights reserved."
    write(*,*) "################################################################"
end subroutine  dateStampPrint

END MODULE io
```

7.6 Result

```
 I am master, I am sending the following data.
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
 I am worker            4  I received the following data from Master
 ################################################################
          Demo code: MPI_PACK created by Wenqiang Feng
 I am worker            1  I received the following data from Master
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
 I am worker            2  I received the following data from Master
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
 I am worker            3  I received the following data from Master
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
               December-12-2015  21:49:01.
 Copyright (c) 2015 WENQIANG FENG. All rights reserved.
 ################################################################
 >>main>> MR timing=    2.9015541076660156E-004  sec on           4  WRs
```

8 MPI_PACK and MPI_UNPACK demo vector format

8.1 Makefile

```
#########################################################################
#      Makefile for demo MPI_PACK and MPI_UNPACK  vector format
#########################################################################
FOR =$(MPI)mpif90 -IMODF -JMODF

# set name of the execution file
EXE = demo

##---------------------> set path to openMPI on local:
# gfortran  on my laptop
#MPI  = /usr/local/bin/
# ifort on my laptop (default one)
#MPI=/opt/intel/compilers_and_libraries_2016.1.150/linux/mpi/intel64/bin/
OUTPUT = OUT/out

# set up the object
OBJ =  OF/io.o OF/mainWR.o OF/mainMR.o OF/main.o

#-==========================create executable: make  ===========#
$(EXE): $(OBJ)
$(FOR) $(OBJ) -o $(EXE)

##-------------------- set code components:--------------------#
OF/io.o: io.f90
$(FOR) -c io.f90 -o OF/io.o

OF/mainWR.o: mainWR.f90
$(FOR) -c mainWR.f90 -o OF/mainWR.o
```

```
OF/mainMR.o: mainMR.f90
$(FOR) -c mainMR.f90 -o OF/mainMR.o

OF/main.o: main.f90
$(FOR) -c main.f90 -o OF/main.o

#------------> lines below a directive MUST start with TAB <-----------#
#----------------------- execute: make run --------------------------#
run:
# @mpirun -np 5 ./$(EXE) < inputdata.dat> $(OUTPUT)
# @mpirun -np 5 ./$(EXE) < inputdata.dat
@mpirun -np 5 ./$(EXE) > $(OUTPUT)
reset:
rm $(EXE) MODF/* OF/* ./*.mod

remove:
rm OUT/*.dat
```

8.2 main.f90

```
!-----------------------------------------------------------
! This demo shows how to use MPI_PACK and MPI_UNPACK in vector format
!
! Author: Wenqiang Feng
!         Department of Mathematics
!         The University of Tennessee
!         Da hu bi
!         Department of Civil Engineering
!         The University of Tennessee
!
! Date  : Dec.8, 2015
!-----------------------------------------------------------
PROGRAM main
USE mainwr
USE mainmr
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
INTEGER :: ierr, nPROC,nWRs, mster, myID
REAL(r8) :: tt0,tt1
!-------------------------------------------------------------------
!       Explanation of variables for MPI  (all integers)
!-------------------------------------------------------------------
!   nPROC    = number of PROCesses = nWRs+1 to use in this run
!   nWRs     = number of workers   = nPROC-1
!   mster    = master rank (=0)
!   myID     = rank of a process (=0,1,2,...,nWRs)
!   Me       = worker's number (rank)  (=1,2,...,nWRs)
!   NodeUP   = rank of neighbor UP from Me
!   NodeDN   = rank of neighbor DN from Me
```

```
!   ierr     = MPI error flag
!-----------------------------------------------------------------

CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE( MPI_COMM_WORLD,NPROC,IERR )
mster = 0
nWRs  = nPROC - 1

Call MPI_COMM_RANK(MPI_COMM_WORLD, MYID, IERR)    !..assigns myID
!
IF( myID .EQ. mster ) THEN
!
tt0 = MPI_WTIME()
CALL MASTER( nWRs )
!
tt1 = MPI_WTIME()
       PRINT*,'>>main>> MR timing= ',tt1-tt0,' sec on ',nWRs,' WRs'
ELSE
       CALL WORKER( nWRs, myID )    !... now MPI is running ...
!
ENDIF
!
CALL MPI_FINALIZE(IERR)
!
END PROGRAM main
```

8.3 mainMR.f90

```
MODULE mainMR
USE io
CONTAINS

SUBROUTINE MASTER(nWRS)
INCLUDE 'mpif.h'
! global parameter
INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n
REAL(r8)  :: a, b

    ! vectors for integer parameters and real parameters, respectively.
INTEGER, ALLOCATABLE, DIMENSION(:)  :: iparms
REAL(r8), ALLOCATABLE, DIMENSION(:) :: parms

! parameters for parallel
INTEGER   :: position
INTEGER, DIMENSION(100)  :: buffer

    Niparms = 1
```

```
    Nparms  = 2
    !
n = 4         ! # of element
    a = 1.0_r8    ! left boundary
b = 4.0_r8   ! right boundary
Print *,'!------------------------------------------------!'
    Print *, 'I am master, I am sending the following data. '
Print *,  ' a=',a,' b=',b,' n=',n
Print *,'!------------------------------------------------!'
! grouping
ALLOCATE(iparms(Niparms), parms(Nparms))
iparms(1:Niparms) = (/n/)
parms(1:Nparms) = (/a,b/)

! packing
position = 0
    call MPI_PACK(parms,Nparms,MPI_DOUBLE_PRECISION,buffer,100,position,MPI_COMM_WORLD,ierr)
    call MPI_PACK(iparms,Niparms,MPI_INTEGER,buffer,100,position,MPI_COMM_WORLD,ierr)

    ! communication
    call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)

DEALLOCATE(iparms,parms)


CALL dateStampPrint
END SUBROUTINE MASTER

END MODULE mainMR
```

8.4 mainWR.f90

```
MODULE mainWR

CONTAINS
SUBROUTINE WORKER(nWRs, myID)
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n
REAL(r8)  :: a, b

! vectors for integer parameters and real parameters, respectively.
INTEGER, DIMENSION(1)  :: iparms
REAL(r8), DIMENSION(2) :: parms

! parameters for parallel
```

```
INTEGER   :: position
INTEGER, DIMENSION(100)  :: buffer
!character(len=100) :: buffer

    Niparms = 1
    Nparms  = 2

! communication
call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)
position = 0

! unpacking
    call MPI_UNPACK(buffer,100,position,parms,Nparms,MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierr)
    call MPI_UNPACK(buffer,100,position,iparms,Niparms,MPI_INTEGER,MPI_COMM_WORLD,ierr)

    a = parms(1)
    b = parms(2)
    n = iparms(1)

    !Print *,'!-----------------------------------------------------!'
Print *, 'I am worker ', myID, ' I received the following data from Master'
Print *,  ' a=',a,' b=',b,' n=',n
!Print *,'!-----------------------------------------------------!'

END SUBROUTINE WORKER
END MODULE mainWR
```

8.5 io.f90

```
MODULE io

CONTAINS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine  dateStampPrint
    integer    :: out_unit
    character(8)  :: date
    character(10) :: time
    character(5)  :: zone
    integer,dimension(8) :: values
    character( len = 9 ), parameter, dimension(12) :: month = (/ &
    'January  ', 'February ', 'March    ', 'April    ', &
    'May      ', 'June     ', 'July     ', 'August   ', &
    'September', 'October  ', 'November ', 'December ' /)

    ! call the interior function
    call date_and_time(date,time,zone,values)
    write(*,*) "################################################################"
    write(*,*) "         Demo code: MPI_PACK created by Wenqiang Feng"
    write (*, '(15x,a,a1,i2,a1,i4,2x,i2,a1,i2.2,a1,i2.2,a1)' ) &
```

```
       trim ( month( values(2))),'-',values(3),'-',values(1),values(5),&
          ':', values(6), ':', values(7), '.'
    write(*,*) "Copyright (c) 2015 WENQIANG FENG. All rights reserved."
    write(*,*) "################################################################"
end subroutine  dateStampPrint

END MODULE io
```

8.6 Result.f90

```
 !-------------------------------------------------!
 I am master, I am sending the following data.
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
 !-------------------------------------------------!
 ############################################################
          Demo code: MPI_PACK created by Wenqiang Feng
               December-12-2015  22:04:26.
 Copyright (c) 2015 WENQIANG FENG. All rights reserved.
 ############################################################
 >>main>> MR timing=    2.4700164794921875E-004  sec on            4  WRs
 I am worker            1  I received the following data from Master
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
 I am worker            2  I received the following data from Master
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
 I am worker            3  I received the following data from Master
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
 I am worker            4  I received the following data from Master
  a=   1.0000000000000000       b=   4.0000000000000000       n=           4
```

9 MPI_SENT and MPI_RECV demo

Figure 13: One dimension's uniform partition for finite element method

9.1 Makefile

```
########################################################################
#             Makefile for demo MPI_SEND and MPI_RECV
########################################################################
FOR =$(MPI)mpif90 -IMODF -JMODF

# set name of the execution file
EXE = demo
```

```
##---------------------> set path to openMPI on local:
# gfortran  on my laptop
#MPI  = /usr/local/bin/
# ifort on my laptop (default one)
#MPI=/opt/intel/compilers_and_libraries_2016.1.150/linux/mpi/intel64/bin/
OUTPUT = OUT/out

# set up the object
OBJ =  OF/setup.o OF/io.o OF/messaging.o OF/mainWR.o OF/mainMR.o OF/main.o

#-==========================create executable: make  ===========#
$(EXE): $(OBJ)
$(FOR) $(OBJ) -o $(EXE)

##-------------------- set code components:--------------------#
OF/io.o: io.f90
$(FOR) -c io.f90 -o OF/io.o

OF/messaging.o: messaging.f90
$(FOR) -c messaging.f90 -o OF/messaging.o

OF/setup.o: setup.f90
$(FOR) -c setup.f90 -o OF/setup.o

OF/mainWR.o: mainWR.f90
$(FOR) -c mainWR.f90 -o OF/mainWR.o

OF/mainMR.o: mainMR.f90
$(FOR) -c mainMR.f90 -o OF/mainMR.o

OF/main.o: main.f90
$(FOR) -c main.f90 -o OF/main.o

#------------> lines below a directive MUST start with TAB <-----------#
#----------------------- execute: make run --------------------------#
run:
@mpirun -np 3 ./$(EXE) < inputdata.dat> $(OUTPUT)
# @mpirun -np 5 ./$(EXE) < inputdata.dat
reset:
rm $(EXE) MODF/* OF/* ./*.mod

remove:
rm OUT/*.dat
```

9.2 main.f90

```
!-----------------------------------------------------------
! This demo shows how to use MPI_PACK and MPI_UNPACK in vector format
!
! Author: Wenqiang Feng
!         Department of Mathematics
```

```
!         The University of Tennessee
!
! Date  : Dec.8, 2015
!-------------------------------------------------------
PROGRAM main
USE mainwr
USE mainmr
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
INTEGER :: ierr, nPROC,nWRs, mster, myID
REAL(r8) :: tt0,tt1
!-------------------------------------------------------------
!       Explanation of variables for MPI  (all integers)
!-------------------------------------------------------------
!   nPROC    = number of PROCesses = nWRs+1 to use in this run
!   nWRs     = number of workers   = nPROC-1
!   mster    = master rank (=0)
!   myID     = rank of a process (=0,1,2,...,nWRs)
!   Me       = worker's number (rank)  (=1,2,...,nWRs)
!   NodeUP   = rank of neighbor UP from Me
!   NodeDN   = rank of neighbor DN from Me
!   ierr     = MPI error flag
!-------------------------------------------------------------

CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE( MPI_COMM_WORLD,NPROC,IERR )
mster = 0
nWRs  = nPROC - 1

Call MPI_COMM_RANK(MPI_COMM_WORLD, MYID, IERR)    !..assigns myID
!
IF( myID .EQ. mster ) THEN
!
tt0 = MPI_WTIME()
CALL MASTER( nWRs )
!
tt1 = MPI_WTIME()
       PRINT*,'>>main>> MR timing= ',tt1-tt0,' sec on ',nWRs,' WRs'
ELSE
       CALL WORKER( nWRs, myID )    !... now MPI is running ...
!
ENDIF
!
CALL MPI_FINALIZE(IERR)
!
END PROGRAM main
```

9.3 mainMR.f90

```
MODULE mainMR
USE io
USE setup
CONTAINS

SUBROUTINE MASTER(nWRS)
INCLUDE 'mpif.h'
! global parameter
INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n
REAL(r8)  :: a, b, dx

    ! vectors for integer parameters and real parameters, respectively.
INTEGER, ALLOCATABLE, DIMENSION(:)  :: iparms
REAL(r8), ALLOCATABLE, DIMENSION(:) :: parms
real(kind=r8), dimension(:), allocatable :: x

! parameters for parallel
INTEGER   :: position
INTEGER, DIMENSION(100)  :: buffer

    Niparms = 1
    Nparms  = 2

! Read run-time parameters from data file, readin in io module.
CALL readin(a,b,n)
!
! set primary parameters
dx = (b-a)/n
allocate(x(0:n+1))

call global_mesh(a, b, dx, n, x)

print *, 'Global x:', x

!
Print *,'!-----------------------------------------------!'
    Print *, 'I am master, I am sending the following data. '
Print *,  ' a=',a,' b=',b,' n=',n
Print *,'!-----------------------------------------------!'
! grouping
ALLOCATE(iparms(Niparms), parms(Nparms))
iparms(1:Niparms) = (/n/)
parms(1:Nparms) = (/a,b/)

! packing
position = 0
    call MPI_PACK(parms,Nparms,MPI_DOUBLE_PRECISION,buffer,100,position,MPI_COMM_WORLD,ierr)
```

```
    call MPI_PACK(iparms,Niparms,MPI_INTEGER,buffer,100,position,MPI_COMM_WORLD,ierr)

    ! communication
    call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)

DEALLOCATE(iparms,parms)


CALL dateStampPrint
END SUBROUTINE MASTER

END MODULE mainMR
```

9.4 mainWR.f90

```
MODULE mainWR
USE setup
USE messaging

CONTAINS
SUBROUTINE WORKER(nWRs, Me)
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n, local_n
REAL(r8)  :: a, b , dx

! vectors for integer parameters and real parameters, respectively.
INTEGER, DIMENSION(1)  :: iparms
REAL(r8), DIMENSION(2) :: parms
real(kind=r8), dimension(:), allocatable :: x, local_U
real(kind=r8), dimension(:), allocatable :: local_x

! parameters for parallel
INTEGER   :: position, NodeUP, NodeDN
INTEGER, DIMENSION(100)  :: buffer

    Niparms = 1
    Nparms  = 2

! communication
call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)
position = 0

! unpacking
    call MPI_UNPACK(buffer,100,position,parms,Nparms,MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierr)
    call MPI_UNPACK(buffer,100,position,iparms,Niparms,MPI_INTEGER,MPI_COMM_WORLD,ierr)
```

```
    a = parms(1)
    b = parms(2)
    n = iparms(1)

    !Print *,'!-----------------------------------------------!'
!Print *, 'I am worker ', Me, ' I received the following data from Master'
!Print *,  ' a=',a,' b=',b,' n=',n
!Print *,'!-----------------------------------------------!'
!
dx = (b-a)/n
local_n = n/nWRs
allocate(x(0:n+1),local_x(0:local_n+1),local_U(0:local_n+1))
    ! generate the global mesh
call global_mesh(a, b, dx, n, x)
!
! generate the local mesh for each worker
local_x = x((Me-1)*local_n:Me*local_n+1)
!
! generate initial value for each worker
call init(Me, local_x, local_U)
!Print *, 'I am worker ', Me, ' I have local U:'
print *, 'local u (before):', local_U
!
NodeUP = Me + 1
NodeDN = Me - 1
call EXCHANGE_BNDRY_MPI( nWRs, Me, NodeUP, NodeDN, local_n, local_U)
print *, 'local u (after):', local_U
END SUBROUTINE WORKER
END MODULE mainWR
```

9.5 messaging.f90

```
module messaging

contains


subroutine EXCHANGE_BNDRY_MPI( nWRs, Me, NodeUP, NodeDN, Mz, U)
INCLUDE 'mpif.h'
!.........Exchange "boundary" values btn neighbors.........!
!..................... every WR does this ...................!
integer, intent(in) :: nWRs, Me, NodeUP, NodeDN, Mz
integer, parameter :: r8 = SELECTED_REAL_KIND(15,307)
integer :: I2, i, Ime, msgtag, status, &
           ierr, msgUP, msgDN, Iup, Iup1
real(kind=r8), dimension(0:), intent(inout) :: U

Iup  = Mz
Iup1 = Mz + 1
```

```
msgUP = 100
msgDN = 200

!.................send bottom row to neighbor down:
if ( Me .ne. 1 ) then
   msgtag = msgDN + Me
   call MPI_SEND(U(1),1,MPI_DOUBLE_PRECISION,NodeDN,msgtag,MPI_COMM_WORLD,ierr)
end if

!.....receive bottom row from neighbor up and save as upper bry:
if ( Me .ne. nWRs ) then
   msgtag = msgDN + NodeUP
   call MPI_RECV(U(Iup1),1,MPI_DOUBLE_PRECISION,NodeUP,msgtag,MPI_COMM_WORLD,status,ierr)
end if

!.................send the top row to neighbor up:
if ( Me .ne. nWRs ) then
   msgtag = msgUP + Me
   call MPI_SEND(U(Iup),1,MPI_DOUBLE_PRECISION,NodeUP,msgtag,MPI_COMM_WORLD,ierr)
end if

!......receive top row from neighbor down and save as lower bry:
if ( Me .ne. 1 ) then
   msgtag = msgUP + NodeDN
   call MPI_RECV(U(0),1,MPI_DOUBLE_PRECISION,NodeDN,msgtag,MPI_COMM_WORLD,status,ierr)
end if

end subroutine EXCHANGE_BNDRY_MPI

end module messaging
```

9.6 setup.f90

```
MODULE setup
CONTAINS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine global_mesh(a, b, dx, M, x)
IMPLICIT NONE
INTEGER, PARAMETER:: r8 = SELECTED_REAL_KIND(15,307)
REAL(KIND=r8), INTENT(IN):: a, b, dx
INTEGER, INTENT(IN):: M
REAL(KIND=r8), DIMENSION(0:), INTENT(OUT):: x
INTEGER:: i

x(0) = a
x(1) = a + 0.5_r8*dx
do i = 2, M
x(i) = x(1) + (i-1)*dx
end do
x(M+1) = b
```

```
end subroutine global_mesh
!---------------------local_mesh-----------------------------
SUBROUTINE mesh(a, b, dx, Mz, x, Me, nWRs)
IMPLICIT NONE
INTEGER, PARAMETER:: r8 = SELECTED_REAL_KIND(15,307)
REAL(KIND=r8), INTENT(IN):: a, b, dx
INTEGER, INTENT(IN):: Mz, Me, nWRs
REAL(KIND=r8), DIMENSION(0:Mz+1), INTENT(OUT):: x
INTEGER:: i

if (Me .eq. 1) then
   x(0) = a
   x(1) = a + 0.5_r8*dx
   do i = 2, Mz+1
      x(i) = x(1) + (i-1)*dx
   end do
else if (Me .eq. nWRs) then
   x(Mz+1) = b
   x(Mz) = b - 0.5_r8*dx
   do i = Mz-1, 0, -1
      x(i) = x(Mz) - (Mz-i)*dx
   end do
else
   x(0) = a + 0.5_r8*dx + ((Me-1)*Mz-1) * dx
   do i = 1, Mz+1
      x(i) = x(0) + i*dx
   end do
end if

END SUBROUTINE mesh
!---------------------Subrountine Init-----------------------------
SUBROUTINE init(Me,x, U)
IMPLICIT NONE
INTEGER, PARAMETER:: r8 = SELECTED_REAL_KIND(15,307)
INTEGER, INTENT(IN):: Me
REAL(KIND=r8), DIMENSION(0:), INTENT(IN):: x
REAL(KIND=r8), DIMENSION(0:), INTENT(OUT):: U
INTEGER :: i

DO i = 0, SIZE(x,1)-1
   U(i) = Me*10_r8+i
END DO

END SUBROUTINE init
END MODULE setup
```

9.7 io.f90

```
MODULE io
```

```
CONTAINS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   SUBROUTINE readin(a, b, n)
   IMPLICIT NONE
!------------------------READ in data-------------------------------
   INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
   INTEGER   :: ierror, n
   REAL(r8)  :: a, b

    ! Read run-time parameters from data file
NAMELIST/inputdata/ a, b, n
!
OPEN(UNIT=75,FILE='inputdata.dat',STATUS='OLD',ACTION='READ',IOSTAT=ierror)
IF(ierror/=0) THEN
  PRINT *,'Error opening input file problemdata.dat. Program stop.'
  STOP
END IF
READ(75,NML=inputdata)
CLOSE(75)
   END SUBROUTINE readin
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine  dateStampPrint
    integer    :: out_unit
    character(8)  :: date
    character(10) :: time
    character(5)  :: zone
    integer,dimension(8) :: values
    character( len = 9 ), parameter, dimension(12) :: month = (/ &
    'January  ', 'February ', 'March    ', 'April    ', &
    'May      ', 'June     ', 'July     ', 'August   ', &
    'September', 'October  ', 'November ', 'December ' /)

    ! call the interior function
    call date_and_time(date,time,zone,values)
    write(*,*) "################################################################"
    write(*,*) "         Demo code: MPI_PACK created by Wenqiang Feng"
    write (*, '(15x,a,a1,i2,a1,i4,2x,i2,a1,i2.2,a1,i2.2,a1)' ) &
       trim ( month( values(2))),'-',values(3),'-',values(1),values(5),&
          ':', values(6), ':', values(7), '.'
    write(*,*) "Copyright (c) 2015 WENQIANG FENG. All rights reserved."
    write(*,*) "################################################################"
end subroutine  dateStampPrint

END MODULE io
```

9.8 inputdata.dat

```
&inputdata
```

```
a = 0.00D-00

b = 4.00D-00

n = 4/
```

9.9 Results

1. 3 processors (2 workers)

```
Global x:   0.0000000000000000       0.50000000000000000        1.5000000000000000
      2.5000000000000000        3.5000000000000000        4.0000000000000000
!--------------------------------------------------!
I am master, I am sending the following data.
 a=   0.0000000000000000       b=   4.0000000000000000       n=           4
!--------------------------------------------------!
################################################################
         Demo code: MPI_PACK created by Wenqiang Feng
local u (before):   20.000000000000000        21.000000000000000
      22.000000000000000        23.000000000000000
local u (before):   10.000000000000000        11.000000000000000
        12.000000000000000        13.000000000000000
              December-12-2015  22:27:55.
Copyright (c) 2015 WENQIANG FENG. All rights reserved.
################################################################
local u (after):   12.000000000000000        21.000000000000000
        22.000000000000000        23.000000000000000
>>main>> MR timing=    1.6093254089355469E-004  sec on            2  WRs
local u (after):   10.000000000000000        11.000000000000000
       12.000000000000000        21.000000000000000
```

2. 5 processors (4 works)

```
Global x:   0.0000000000000000       0.50000000000000000        1.5000000000000000
        2.5000000000000000        3.5000000000000000        4.0000000000000000
!--------------------------------------------------!
I am master, I am sending the following data.
 a=   0.0000000000000000       b=   4.0000000000000000       n=           4
!--------------------------------------------------!
local u (before):   30.000000000000000        31.000000000000000        32.000000000000000
################################################################
         Demo code: MPI_PACK created by Wenqiang Feng
              December-12-2015  22:42:18.
Copyright (c) 2015 WENQIANG FENG. All rights reserved.
################################################################
local u (before):   10.000000000000000        11.000000000000000        12.000000000000000
local u (after):   10.000000000000000        11.000000000000000        21.000000000000000
local u (before):   20.000000000000000        21.000000000000000        22.000000000000000
local u (after):   11.000000000000000        21.000000000000000        31.000000000000000
local u (after):   21.000000000000000        31.000000000000000        41.000000000000000
local u (before):   40.000000000000000        41.000000000000000        42.000000000000000
```

```
local u (after):   31.000000000000000        41.000000000000000        42.000000000000000
>>main>> MR timing=    3.8194656372070312E-004  sec on            4  WRs
```

10 MPI_Barrier and Collective Communication without Boundary Points

This example shows how to use the call MPI_Barrier that allows to synchronize processes. When a process encounters an MPI_Barrier call, it waits for all the other processes of the given communicator to reach the same point. Nevertheless, the function MPI_Barrier is essential in many other situations.

One use of MPI_Barrier is for example to control access to an external resource such as the filesystem, which is not accessed using MPI. For example, if you want each process to write stuff to a file in sequence, or return a results to master.

10.1 Makefile

```
########################################################################
#             Makefile for demo MPI_Barrier
########################################################################
FOR =$(MPI)mpif90 -IMODF -JMODF

# set name of the execution file
EXE = demo

##---------------------> set path to openMPI on local:
# gfortran  on my laptop
#MPI  = /usr/local/bin/
# ifort on my laptop (default one)
#MPI=/opt/intel/compilers_and_libraries_2016.1.150/linux/mpi/intel64/bin/
OUTPUT = OUT/out

# set up the object
OBJ =  OF/setup.o OF/io.o OF/messaging.o OF/mainWR.o OF/mainMR.o OF/main.o

#-==========================create executable: make  ===========#
$(EXE): $(OBJ)
$(FOR) $(OBJ) -o $(EXE)

##-------------------- set code components:----------------------#
OF/io.o: io.f90
$(FOR) -c io.f90 -o OF/io.o

OF/messaging.o: messaging.f90
$(FOR) -c messaging.f90 -o OF/messaging.o

OF/setup.o: setup.f90
$(FOR) -c setup.f90 -o OF/setup.o

OF/mainWR.o: mainWR.f90
$(FOR) -c mainWR.f90 -o OF/mainWR.o
```

```
OF/mainMR.o: mainMR.f90
$(FOR) -c mainMR.f90 -o OF/mainMR.o

OF/main.o: main.f90
$(FOR) -c main.f90 -o OF/main.o

#------------> lines below a directive MUST start with TAB <-----------#
#----------------------- execute: make run --------------------------#
run:
# @mpirun -np 3 ./$(EXE) < inputdata.dat> $(OUTPUT)
@mpirun -np 5 ./$(EXE) < inputdata.dat
reset:
rm $(EXE) MODF/* OF/* ./*.mod

remove:
rm OUT/*.dat
```

10.2 main.f90

```
!-----------------------------------------------------------
! This demo shows how to use MPI_PACK and MPI_UNPACK in vector format
!
! Author: Wenqiang Feng
!         Department of Mathematics
!         The University of Tennessee
!
! Date  : Dec.8, 2015
!-----------------------------------------------------------
PROGRAM main
USE mainwr
USE mainmr
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
INTEGER :: ierr, nPROC,nWRs, mster, myID
REAL(kind=r8) :: tt0,tt1
!-------------------------------------------------------------------
!       Explanation of variables for MPI  (all integers)
!-------------------------------------------------------------------
!   nPROC    = number of PROCesses = nWRs+1 to use in this run
!   nWRs     = number of workers   = nPROC-1
!   mster    = master rank (=0)
!   myID     = rank of a process (=0,1,2,...,nWRs)
!   Me       = worker's number (rank)  (=1,2,...,nWRs)
!   NodeUP   = rank of neighbor UP from Me
!   NodeDN   = rank of neighbor DN from Me
!   ierr     = MPI error flag
!-------------------------------------------------------------------

CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE( MPI_COMM_WORLD,NPROC,IERR )
```

```
mster = 0
nWRs  = nPROC - 1

Call MPI_COMM_RANK(MPI_COMM_WORLD, MYID, IERR)    !..assigns myID
!
IF( myID .EQ. mster ) THEN
!
tt0 = MPI_WTIME()
CALL MASTER( nWRs )
!
tt1 = MPI_WTIME()
       PRINT*,'>>main>> MR timing= ',tt1-tt0,' sec on ',nWRs,' WRs'
ELSE
       CALL WORKER( nWRs, myID )    !... now MPI is running ...
!
ENDIF
!
CALL MPI_FINALIZE(IERR)
!
END PROGRAM main
```

10.3 mainMR.f90

```
MODULE mainMR
USE io
USE setup
USE messaging , ONLY: RECV_OUTPUT_MPI
CONTAINS

SUBROUTINE MASTER(nWRS)
INCLUDE 'mpif.h'
! global parameter
INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n
REAL(kind=r8)  :: a, b, dx

    ! vectors for integer parameters and real parameters, respectively.
INTEGER, ALLOCATABLE, DIMENSION(:)  :: iparms
REAL(kind=r8), ALLOCATABLE, DIMENSION(:) :: parms
real(kind=r8), dimension(:), allocatable :: x, U

! parameters for parallel
INTEGER   :: position
INTEGER, DIMENSION(100)  :: buffer

    Niparms = 1
    Nparms  = 2
```

```
! Read run-time parameters from data file, readin in io module.
CALL readin(a,b,n)
!
! set primary parameters
dx = (b-a)/n
allocate(x(0:n+1),U(0:n+1))

call global_mesh(a, b, dx, n, x)

print *, 'Global x:', x

!
Print *,'!-------------------------------------------------!'
    Print *, 'I am master, I am sending the following data. '
Print *,  ' a=',a,' b=',b,' n=',n
Print *,'!-------------------------------------------------!'
! grouping
ALLOCATE(iparms(Niparms), parms(Nparms))
iparms(1:Niparms) = (/n/)
parms(1:Nparms) = (/a,b/)

! packing
position = 0
    call MPI_PACK(parms,Nparms,MPI_DOUBLE_PRECISION,buffer,100,position,MPI_COMM_WORLD,ierr)
    call MPI_PACK(iparms,Niparms,MPI_INTEGER,buffer,100,position,MPI_COMM_WORLD,ierr)

    ! communication
    call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)
    ! set up the barrier
call MPI_Barrier(MPI_COMM_WORLD,ierr)
! Collecte data from each worker
CALL RECV_OUTPUT_MPI(nWRs, n, U)

print *, 'collect',U(1:n)
CALL dateStampPrint
DEALLOCATE(iparms,parms)
END SUBROUTINE MASTER

END MODULE mainMR
```

10.4 mainWR.f90

```
MODULE mainWR
USE setup
USE messaging

CONTAINS
SUBROUTINE WORKER(nWRs, Me)
INCLUDE 'mpif.h'
```

```
INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n, local_n
REAL(kind=r8)  :: a, b , dx

! vectors for integer parameters and real parameters, respectively.
INTEGER, DIMENSION(1)  :: iparms
REAL(kind=r8), DIMENSION(2) :: parms
real(kind=r8), dimension(:), allocatable :: x, local_U
real(kind=r8), dimension(:), allocatable :: local_x

! parameters for parallel
INTEGER   :: position, NodeUP, NodeDN
INTEGER, DIMENSION(100)  :: buffer

    Niparms = 1
    Nparms  = 2

! communication
call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)
position = 0

! unpacking
    call MPI_UNPACK(buffer,100,position,parms,Nparms,MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierr)
    call MPI_UNPACK(buffer,100,position,iparms,Niparms,MPI_INTEGER,MPI_COMM_WORLD,ierr)

    a = parms(1)
    b = parms(2)
    n = iparms(1)

    !Print *,'!-------------------------------------------------!'
!Print *, 'I am worker ', Me, ' I received the following data from Master'
!Print *,  ' a=',a,' b=',b,' n=',n
!Print *,'!-------------------------------------------------!'
!
dx = (b-a)/n
local_n = n/nWRs
allocate(x(0:n+1),local_x(0:local_n+1),local_U(0:local_n+1))
    ! generate the global mesh
call global_mesh(a, b, dx, n, x)
!
! generate the local mesh for each worker
local_x = x((Me-1)*local_n:Me*local_n+1)
!
! generate initial value for each worker
call init(Me, local_x, local_U)
!Print *, 'I am worker ', Me, ' I have local U:'
print *, 'local u (before):', local_U
!
```

```
NodeUP = Me + 1
NodeDN = Me - 1
call EXCHANGE_BNDRY_MPI( nWRs, Me, NodeUP, NodeDN, local_n, local_U)
print *, 'local u (after):', local_U

! set up barrier
call MPI_Barrier(MPI_COMM_WORLD,ierr)
! send data to master
CALL SEND_OUTPUT_MPI(Me,local_n,local_U)
END SUBROUTINE WORKER
END MODULE mainWR
```

10.5 messaging.f90

```
module messaging

contains
!---------------------boundary points exahange--------------------------
subroutine EXCHANGE_BNDRY_MPI( nWRs, Me, NodeUP, NodeDN, Mz, U)
INCLUDE 'mpif.h'
!.........Exchange "boundary" values btn neighbors.........!
!..................... every WR does this ...................!
integer, intent(in) :: nWRs, Me, NodeUP, NodeDN, Mz
integer, parameter :: r8 = SELECTED_REAL_KIND(15,307)
integer :: I2, i, Ime, msgtag, status, &
           ierr, msgUP, msgDN, Iup, Iup1
real(kind=r8), dimension(0:), intent(inout) :: U

Iup  = Mz
Iup1 = Mz + 1
msgUP = 100
msgDN = 200

!.................send bottom row to neighbor down:
if ( Me .ne. 1 ) then
   msgtag = msgDN + Me
   call MPI_SEND(U(1),1,MPI_DOUBLE_PRECISION,NodeDN,msgtag,MPI_COMM_WORLD,ierr)
end if

!.....receive bottom row from neighbor up and save as upper bry:
if ( Me .ne. nWRs ) then
   msgtag = msgDN + NodeUP
   call MPI_RECV(U(Iup1),1,MPI_DOUBLE_PRECISION,NodeUP,msgtag,MPI_COMM_WORLD,status,ierr)
end if

!.................send the top row to neighbor up:
if ( Me .ne. nWRs ) then
   msgtag = msgUP + Me
   call MPI_SEND(U(Iup),1,MPI_DOUBLE_PRECISION,NodeUP,msgtag,MPI_COMM_WORLD,ierr)
end if
```

```
!......receive top row from neighbor down and save as lower bry:
if ( Me .ne. 1 ) then
   msgtag = msgUP + NodeDN
   call MPI_RECV(U(0),1,MPI_DOUBLE_PRECISION,NodeDN,msgtag,MPI_COMM_WORLD,status,ierr)
end if

end subroutine EXCHANGE_BNDRY_MPI

!----------------collect the data from each worker----------------------
SUBROUTINE RECV_OUTPUT_MPI(nWRS,n,U)
!-----------------------only MR does this -----------------------------!
   INCLUDE "mpif.h"

   INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
   INTEGER  :: nWRs, n, Ln,I2,jme,msgtag,ierr,i
   REAL(kind=r8) :: U(0:n+1)

   ! set local n
   Ln = n / nWRs

   DO i = 1, nWRs
I2  = Ln
        Jme    = (i-1)*Ln+1
        msgtag = 1000 + i
CALL MPI_RECV(U(Jme),I2,MPI_DOUBLE_PRECISION,MPI_ANY_SOURCE,&
               msgtag,MPI_COMM_WORLD,MPI_STATUS_IGNORE,ierr)
   ENDDO

   RETURN

END SUBROUTINE RECV_OUTPUT_MPI

!----------------sent the data  to master-----------------------------
SUBROUTINE SEND_OUTPUT_MPI(Me, Ln, U)
   INCLUDE "mpif.h"

   INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
   INTEGER  :: Me, Ln, I2, mster, msgtag, ierr
   REAL(kind=r8) :: U(0:Ln+1)

   mster  = 0
   I2     = Ln
   msgtag = 1000 + Me
   CALL MPI_SEND(U(1),I2,MPI_DOUBLE_PRECISION,mster,msgtag, &
MPI_COMM_WORLD,ierr)

   ! Return
   RETURN

   END SUBROUTINE SEND_OUTPUT_MPI
```

```
end module messaging
```

10.6 setup.f90

```
MODULE setup
CONTAINS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine global_mesh(a, b, dx, M, x)
IMPLICIT NONE
INTEGER, PARAMETER:: r8 = SELECTED_REAL_KIND(15,307)
REAL(KIND=r8), INTENT(IN):: a, b, dx
INTEGER, INTENT(IN):: M
REAL(KIND=r8), DIMENSION(0:), INTENT(OUT):: x
INTEGER:: i

x(0) = a
x(1) = a + 0.5_r8*dx
do i = 2, M
x(i) = x(1) + (i-1)*dx
end do
x(M+1) = b

end subroutine global_mesh
!---------------------local_mesh-----------------------------
SUBROUTINE mesh(a, b, dx, Mz, x, Me, nWRs)
IMPLICIT NONE
INTEGER, PARAMETER:: r8 = SELECTED_REAL_KIND(15,307)
REAL(KIND=r8), INTENT(IN):: a, b, dx
INTEGER, INTENT(IN):: Mz, Me, nWRs
REAL(KIND=r8), DIMENSION(0:Mz+1), INTENT(OUT):: x
INTEGER:: i

if (Me .eq. 1) then
   x(0) = a
   x(1) = a + 0.5_r8*dx
   do i = 2, Mz+1
      x(i) = x(1) + (i-1)*dx
   end do
else if (Me .eq. nWRs) then
   x(Mz+1) = b
   x(Mz) = b - 0.5_r8*dx
   do i = Mz-1, 0, -1
      x(i) = x(Mz) - (Mz-i)*dx
   end do
else
   x(0) = a + 0.5_r8*dx + ((Me-1)*Mz-1) * dx
   do i = 1, Mz+1
      x(i) = x(0) + i*dx
   end do
```

```
end if

END SUBROUTINE mesh
!---------------------Subrountine Init-----------------------------
SUBROUTINE init(Me,x, U)
IMPLICIT NONE
INTEGER, PARAMETER:: r8 = SELECTED_REAL_KIND(15,307)
INTEGER, INTENT(IN):: Me
REAL(KIND=r8), DIMENSION(0:), INTENT(IN):: x
REAL(KIND=r8), DIMENSION(0:), INTENT(OUT):: U
INTEGER :: i

DO i = 0, SIZE(x,1)-1
   U(i) = Me*10_r8+i
END DO

END SUBROUTINE init
END MODULE setup
```

10.7 io.f90

```
MODULE io

CONTAINS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   SUBROUTINE readin(a, b, n)
   IMPLICIT NONE
!------------------------READ in data-------------------------------
   INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
   INTEGER   :: ierror, n
   REAL(kind=r8)  :: a, b

    ! Read run-time parameters from data file
NAMELIST/inputdata/ a, b, n
!
OPEN(UNIT=75,FILE='inputdata.dat',STATUS='OLD',ACTION='READ',IOSTAT=ierror)
IF(ierror/=0) THEN
  PRINT *,'Error opening input file problemdata.dat. Program stop.'
  STOP
END IF
READ(75,NML=inputdata)
CLOSE(75)
   END SUBROUTINE readin
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine  dateStampPrint
    integer    :: out_unit
    character(8)  :: date
    character(10) :: time
    character(5)  :: zone
```

```
    integer,dimension(8) :: values
    character( len = 9 ), parameter, dimension(12) :: month = (/ &
    'January  ', 'February ', 'March    ', 'April    ', &
    'May      ', 'June     ', 'July     ', 'August   ', &
    'September', 'October  ', 'November ', 'December ' /)

    ! call the interior function
    call date_and_time(date,time,zone,values)
    write(*,*) "################################################################"
    write(*,*) "         Demo code: MPI_PACK created by Wenqiang Feng"
    write (*, '(15x,a,a1,i2,a1,i4,2x,i2,a1,i2.2,a1,i2.2,a1)' ) &
       trim ( month( values(2))),'-',values(3),'-',values(1),values(5),&
          ':', values(6), ':', values(7), '.'
    write(*,*) "Copyright (c) 2015 WENQIANG FENG. All rights reserved."
    write(*,*) "################################################################"
end subroutine  dateStampPrint

END MODULE io
```

10.8 inputdata.dat

```
&inputdata

a = 0.00D-00

b = 4.00D-00

n = 4/
```

10.9 Results

1. Result with 3 processor (2 workers)

```
Global x:   0.0000000000000000       0.50000000000000000        1.5000000000000000
        2.5000000000000000        3.5000000000000000        4.0000000000000000
!------------------------------------------------!
I am master, I am sending the following data.
 a=   0.0000000000000000       b=   4.0000000000000000       n=           4
!------------------------------------------------!
local u (before):   10.000000000000000        11.000000000000000
       12.000000000000000        13.000000000000000
local u (before):   20.000000000000000        21.000000000000000
        22.000000000000000        23.000000000000000
local u (after):   12.000000000000000        21.000000000000000
       22.000000000000000        23.000000000000000
local u (after):   10.000000000000000        11.000000000000000
      12.000000000000000        21.000000000000000
collect   11.000000000000000        12.000000000000000        21.000000000000000
       22.000000000000000
################################################################
         Demo code: MPI_PACK created by Wenqiang Feng
```

```
               December-13-2015  21:44:01.
 Copyright (c) 2015 WENQIANG FENG. All rights reserved.
 ###############################################################
 >>main>> MR timing=    2.3007392883300781E-004  sec on            2  WRs
```

2. Result with 5 processors (4 workers)

```
Global x:   0.0000000000000000       0.50000000000000000        1.5000000000000000
      2.5000000000000000        3.5000000000000000        4.0000000000000000
 !-------------------------------------------------!
 I am master, I am sending the following data.
  a=   0.0000000000000000       b=   4.0000000000000000       n=           4
 !-------------------------------------------------!
 local u (before):   30.000000000000000        31.000000000000000
       32.000000000000000
 local u (before):   10.000000000000000        11.000000000000000
       12.000000000000000
 local u (after):   10.000000000000000        11.000000000000000
        21.000000000000000
 local u (before):   20.000000000000000        21.000000000000000
        22.000000000000000
 local u (after):   11.000000000000000        21.000000000000000
       31.000000000000000
 local u (after):   21.000000000000000        31.000000000000000
        41.000000000000000
 local u (before):   40.000000000000000        41.000000000000000
      42.000000000000000
 local u (after):   31.000000000000000        41.000000000000000
       42.000000000000000
 collected data   11.000000000000000        21.000000000000000
       31.000000000000000        41.000000000000000
 ###############################################################
          Demo code: MPI_PACK created by Wenqiang Feng
               December-13-2015  21:46:32.
 Copyright (c) 2015 WENQIANG FENG. All rights reserved.
 ###############################################################
 >>main>> MR timing=    5.8102607727050781E-004  sec on            4  WRs
```

11 MPI_Barrier and Collective Communication with Boundary Points

11.1 Makefile

```
#######################################################################
#             Makefile for demo MPI_Barrier
#######################################################################
FOR =$(MPI)mpif90 -IMODF -JMODF

# set name of the execution file
EXE = demo
```

```
##---------------------> set path to openMPI on local:
# gfortran  on my laptop
#MPI  = /usr/local/bin/
# ifort on my laptop (default one)
#MPI=/opt/intel/compilers_and_libraries_2016.1.150/linux/mpi/intel64/bin/
OUTPUT = OUT/out

# set up the object
OBJ =  OF/setup.o OF/io.o OF/messaging.o OF/mainWR.o OF/mainMR.o OF/main.o

#-==========================create executable: make  ===========#
$(EXE): $(OBJ)
$(FOR) $(OBJ) -o $(EXE)

##-------------------- set code components:--------------------#
OF/io.o: io.f90
$(FOR) -c io.f90 -o OF/io.o

OF/messaging.o: messaging.f90
$(FOR) -c messaging.f90 -o OF/messaging.o

OF/setup.o: setup.f90
$(FOR) -c setup.f90 -o OF/setup.o

OF/mainWR.o: mainWR.f90
$(FOR) -c mainWR.f90 -o OF/mainWR.o

OF/mainMR.o: mainMR.f90
$(FOR) -c mainMR.f90 -o OF/mainMR.o

OF/main.o: main.f90
$(FOR) -c main.f90 -o OF/main.o

#------------> lines below a directive MUST start with TAB <-----------#
#----------------------- execute: make run --------------------------#
run:
# @mpirun -np 3 ./$(EXE) < inputdata.dat> $(OUTPUT)
@mpirun -np 5 ./$(EXE) < inputdata.dat
reset:
rm $(EXE) MODF/* OF/* ./*.mod

remove:
rm OUT/*.dat
```

11.2 main.f90

```
!-----------------------------------------------------------
! This demo shows how to use MPI_PACK and MPI_UNPACK in vector format
!
! Author: Wenqiang Feng
!         Department of Mathematics
```

```
!         The University of Tennessee
!
! Date  : Dec.8, 2015
!-------------------------------------------------------
PROGRAM main
USE mainwr
USE mainmr
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
INTEGER :: ierr, nPROC,nWRs, mster, myID
REAL(kind=r8) :: tt0,tt1
!-----------------------------------------------------------------
!       Explanation of variables for MPI  (all integers)
!-----------------------------------------------------------------
!   nPROC    = number of PROCesses = nWRs+1 to use in this run
!   nWRs     = number of workers   = nPROC-1
!   mster    = master rank (=0)
!   myID     = rank of a process (=0,1,2,...,nWRs)
!   Me       = worker's number (rank)  (=1,2,...,nWRs)
!   NodeUP   = rank of neighbor UP from Me
!   NodeDN   = rank of neighbor DN from Me
!   ierr     = MPI error flag
!-----------------------------------------------------------------

CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE( MPI_COMM_WORLD,NPROC,IERR )
mster = 0
nWRs  = nPROC - 1

Call MPI_COMM_RANK(MPI_COMM_WORLD, MYID, IERR)    !..assigns myID
!
IF( myID .EQ. mster ) THEN
!
tt0 = MPI_WTIME()
CALL MASTER( nWRs )
!
tt1 = MPI_WTIME()
       PRINT*,'>>main>> MR timing= ',tt1-tt0,' sec on ',nWRs,' WRs'
ELSE
       CALL WORKER( nWRs, myID )    !... now MPI is running ...
!
ENDIF
!
CALL MPI_FINALIZE(IERR)
!
END PROGRAM main
```

11.3 mainMR.f90

```
MODULE mainMR
USE io
USE setup
USE messaging , ONLY: RECV_OUTPUT_MPI
CONTAINS

SUBROUTINE MASTER(nWRS)
INCLUDE 'mpif.h'
! global parameter
INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n
REAL(kind=r8)  :: a, b, dx

    ! vectors for integer parameters and real parameters, respectively.
INTEGER, ALLOCATABLE, DIMENSION(:)  :: iparms
REAL(kind=r8), ALLOCATABLE, DIMENSION(:) :: parms
real(kind=r8), dimension(:), allocatable :: x, U

! parameters for parallel
INTEGER   :: position
INTEGER, DIMENSION(100)  :: buffer

    Niparms = 1
    Nparms  = 2

! Read run-time parameters from data file, readin in io module.
CALL readin(a,b,n)
!
! set primary parameters
dx = (b-a)/n
allocate(x(0:n+1),U(0:n+1))

call global_mesh(a, b, dx, n, x)

print *, 'Global x:', x

!
Print *,'!-------------------------------------------------!'
    Print *, 'I am master, I am sending the following data. '
Print *,  ' a=',a,' b=',b,' n=',n
Print *,'!-------------------------------------------------!'
! grouping
ALLOCATE(iparms(Niparms), parms(Nparms))
iparms(1:Niparms) = (/n/)
parms(1:Nparms) = (/a,b/)

! packing
position = 0
```

```
    call MPI_PACK(parms,Nparms,MPI_DOUBLE_PRECISION,buffer,100,position,MPI_COMM_WORLD,ierr)
    call MPI_PACK(iparms,Niparms,MPI_INTEGER,buffer,100,position,MPI_COMM_WORLD,ierr)

    ! communication
    call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)

    ! set up the barrier
call MPI_Barrier(MPI_COMM_WORLD,ierr)

! collect the data from each worker
CALL RECV_OUTPUT_MPI(nWRs, n, U)

print *, 'collect',U
CALL dateStampPrint
DEALLOCATE(iparms,parms)
END SUBROUTINE MASTER

END MODULE mainMR
```

11.4 mainWR.f90

```
MODULE mainWR
USE setup
USE messaging

CONTAINS
SUBROUTINE WORKER(nWRs, Me)
INCLUDE 'mpif.h'

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)

! Parameters which need to be packed
INTEGER   :: Niparms, Nparms, n, local_n
REAL(kind=r8)  :: a, b , dx

! vectors for integer parameters and real parameters, respectively.
INTEGER, DIMENSION(1)  :: iparms
REAL(kind=r8), DIMENSION(2) :: parms
real(kind=r8), dimension(:), allocatable :: x, local_U
real(kind=r8), dimension(:), allocatable :: local_x

! parameters for parallel
INTEGER   :: position, NodeUP, NodeDN
INTEGER, DIMENSION(100)  :: buffer

    Niparms = 1
    Nparms  = 2

! communication
```

```
call MPI_BCAST(buffer,100,MPI_PACKED,0,MPI_COMM_WORLD,ierr)
position = 0

! unpacking
    call MPI_UNPACK(buffer,100,position,parms,Nparms,MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierr)
    call MPI_UNPACK(buffer,100,position,iparms,Niparms,MPI_INTEGER,MPI_COMM_WORLD,ierr)

    a = parms(1)
    b = parms(2)
    n = iparms(1)

    !Print *,'!-----------------------------------------------------!'
!Print *, 'I am worker ', Me, ' I received the following data from Master'
!Print *,  ' a=',a,' b=',b,' n=',n
!Print *,'!-----------------------------------------------------!'
!
dx = (b-a)/n
local_n = n/nWRs
allocate(x(0:n+1),local_x(0:local_n+1),local_U(0:local_n+1))
    ! generate the global mesh
call global_mesh(a, b, dx, n, x)
!
! generate the local mesh for each worker
local_x = x((Me-1)*local_n:Me*local_n+1)
!
! generate initial value for each worker
call init(Me, local_x, local_U)
!Print *, 'I am worker ', Me, ' I have local U:'
print *, 'local u (before):', local_U
!
NodeUP = Me + 1
NodeDN = Me - 1
call EXCHANGE_BNDRY_MPI( nWRs, Me, NodeUP, NodeDN, local_n, local_U)
print *, 'local u (after):', local_U

! set up the barrier
call MPI_Barrier(MPI_COMM_WORLD,ierr)

! send data to master
CALL SEND_OUTPUT_MPI(Me, nWRs, local_n,local_U)
END SUBROUTINE WORKER
END MODULE mainWR
```

11.5 messaging.f90

```
module messaging

contains
!---------------------boundary points exahange--------------------------
subroutine EXCHANGE_BNDRY_MPI( nWRs, Me, NodeUP, NodeDN, Mz, U)
```

```
INCLUDE 'mpif.h'
!..........Exchange "boundary" values btn neighbors.........!
!..................... every WR does this ...................!
integer, intent(in) :: nWRs, Me, NodeUP, NodeDN, Mz
integer, parameter :: r8 = SELECTED_REAL_KIND(15,307)
integer :: I2, i, Ime, msgtag, status, &
           ierr, msgUP, msgDN, Iup, Iup1
real(kind=r8), dimension(0:), intent(inout) :: U

Iup  = Mz
Iup1 = Mz + 1
msgUP = 100
msgDN = 200

!.................send bottom row to neighbor down:
if ( Me .ne. 1 ) then
   msgtag = msgDN + Me
   call MPI_SEND(U(1),1,MPI_DOUBLE_PRECISION,NodeDN,msgtag,MPI_COMM_WORLD,ierr)
end if

!.....receive bottom row from neighbor up and save as upper bry:
if ( Me .ne. nWRs ) then
   msgtag = msgDN + NodeUP
   call MPI_RECV(U(Iup1),1,MPI_DOUBLE_PRECISION,NodeUP,msgtag,MPI_COMM_WORLD,status,ierr)
end if

!.................send the top row to neighbor up:
if ( Me .ne. nWRs ) then
   msgtag = msgUP + Me
   call MPI_SEND(U(Iup),1,MPI_DOUBLE_PRECISION,NodeUP,msgtag,MPI_COMM_WORLD,ierr)
end if

!......receive top row from neighbor down and save as lower bry:
if ( Me .ne. 1 ) then
   msgtag = msgUP + NodeDN
   call MPI_RECV(U(0),1,MPI_DOUBLE_PRECISION,NodeDN,msgtag,MPI_COMM_WORLD,status,ierr)
end if

end subroutine EXCHANGE_BNDRY_MPI

!----------------collect the data from each worker----------------------
SUBROUTINE RECV_OUTPUT_MPI(nWRS,n,U)
!-----------------------only MR does this -----------------------------!
   INCLUDE "mpif.h"

   INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
   INTEGER  :: nWRs, n, local_n,I2,jme,msgtag,ierr,i
   REAL(kind=r8) :: U(0:n+1)

   ! set local n
   local_n = n / nWRs
```

```
if (nWRs .eq. 1) then
msgtag = 1001
I2 = local_n + 2
J  = i
call MPI_RECV(U(0),I2,MPI_DOUBLE_PRECISION, J, &
                msgtag,MPI_COMM_WORLD,MPI_STATUS_IGNORE,ierr)
return
end if
!
DO i= 1, nWRs
!
if (i .eq. 1) then
! left segment
I2 = local_n+1
msgtag = 1000 + i
J = i
call MPI_RECV(U(0),I2,MPI_DOUBLE_PRECISION,J,&
                 msgtag,MPI_COMM_WORLD,MPI_STATUS_IGNORE,ierr)
else if (i .eq. nWRs) then
! right segment
I2 =local_n+1
J  = i
Ime = (i-1)* local_n + 1
msgtag = 1000 + i
call MPI_RECV(U(Ime),I2,MPI_DOUBLE_PRECISION,J, &
                msgtag,MPI_COMM_WORLD,MPI_STATUS_IGNORE,ierr)
else
! inner segment
I2 = local_n
J  = i
Ime = ( i - 1 ) * local_n + 1
msgtag = 1000 + i
call MPI_RECV(U(Ime),I2,MPI_DOUBLE_PRECISION,J,&
                msgtag,MPI_COMM_WORLD,MPI_STATUS_IGNORE,ierr)
end if


END DO

   RETURN

END SUBROUTINE RECV_OUTPUT_MPI

!----------------sent the data  to master-----------------------------
SUBROUTINE SEND_OUTPUT_MPI(Me, nWRs, local_n, U)
INCLUDE "mpif.h"

INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
INTEGER  :: Me, nWRs, local_n, I2, mster, msgtag, ierr
REAL(kind=r8) :: U(0:local_n+1)
```

```
mster  = 0
I2     = local_n

if (nWRs .eq. 1) then
msgtag = 1001
I2 = local_n + 2
call MPI_SEND(U(0),I2,MPI_DOUBLE_PRECISION,mster,msgtag,MPI_COMM_WORLD,ierr)
return
end if

if (Me .eq. 1) then
! bottom segment
I2 = local_n+1
msgtag = 1000 + Me
call MPI_SEND(U(0),I2,MPI_DOUBLE_PRECISION,mster,msgtag,MPI_COMM_WORLD,ierr)
else if (Me .eq. nWRs) then
! top segment
I2 =local_n+1
msgtag = 1000 + Me
 !  print *, 'local _n ', local_n
call MPI_SEND(U(1),I2,MPI_DOUBLE_PRECISION,mster,msgtag,MPI_COMM_WORLD,ierr)
else
! inner segment
I2 = local_n
msgtag = 1000 + Me
call MPI_SEND(U(1),I2,MPI_DOUBLE_PRECISION,mster,msgtag,MPI_COMM_WORLD,ierr)
end if


!   CALL MPI_SEND(U(1),I2,MPI_DOUBLE_PRECISION,mster,msgtag, &
! MPI_COMM_WORLD,ierr)

   ! Return
   RETURN

   END SUBROUTINE SEND_OUTPUT_MPI

end module messaging
```

11.6 setup.f90

```
MODULE setup
CONTAINS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine global_mesh(a, b, dx, M, x)
IMPLICIT NONE
INTEGER, PARAMETER:: r8 = SELECTED_REAL_KIND(15,307)
REAL(KIND=r8), INTENT(IN):: a, b, dx
INTEGER, INTENT(IN):: M
```

```
REAL(KIND=r8), DIMENSION(0:), INTENT(OUT):: x
INTEGER:: i

x(0) = a
x(1) = a + 0.5_r8*dx
do i = 2, M
x(i) = x(1) + (i-1)*dx
end do
x(M+1) = b

end subroutine global_mesh
!---------------------local_mesh-----------------------------
SUBROUTINE mesh(a, b, dx, Mz, x, Me, nWRs)
IMPLICIT NONE
INTEGER, PARAMETER:: r8 = SELECTED_REAL_KIND(15,307)
REAL(KIND=r8), INTENT(IN):: a, b, dx
INTEGER, INTENT(IN):: Mz, Me, nWRs
REAL(KIND=r8), DIMENSION(0:Mz+1), INTENT(OUT):: x
INTEGER:: i

if (Me .eq. 1) then
   x(0) = a
   x(1) = a + 0.5_r8*dx
   do i = 2, Mz+1
      x(i) = x(1) + (i-1)*dx
   end do
else if (Me .eq. nWRs) then
   x(Mz+1) = b
   x(Mz) = b - 0.5_r8*dx
   do i = Mz-1, 0, -1
      x(i) = x(Mz) - (Mz-i)*dx
   end do
else
   x(0) = a + 0.5_r8*dx + ((Me-1)*Mz-1) * dx
   do i = 1, Mz+1
      x(i) = x(0) + i*dx
   end do
end if

END SUBROUTINE mesh
!---------------------Subrountine Init-----------------------------
SUBROUTINE init(Me,x, U)
IMPLICIT NONE
INTEGER, PARAMETER:: r8 = SELECTED_REAL_KIND(15,307)
INTEGER, INTENT(IN):: Me
REAL(KIND=r8), DIMENSION(0:), INTENT(IN):: x
REAL(KIND=r8), DIMENSION(0:), INTENT(OUT):: U
INTEGER :: i

DO i = 0, SIZE(x,1)-1
   U(i) = Me*10_r8+i
```

```
END DO

END SUBROUTINE init
END MODULE setup
```

11.7 io.f90

```
MODULE io

CONTAINS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   SUBROUTINE readin(a, b, n)
   IMPLICIT NONE
!-----------------------READ in data-------------------------------
   INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,307)
   INTEGER   :: ierror, n
   REAL(kind=r8)  :: a, b

    ! Read run-time parameters from data file
NAMELIST/inputdata/ a, b, n
!
OPEN(UNIT=75,FILE='inputdata.dat',STATUS='OLD',ACTION='READ',IOSTAT=ierror)
IF(ierror/=0) THEN
  PRINT *,'Error opening input file problemdata.dat. Program stop.'
  STOP
END IF
READ(75,NML=inputdata)
CLOSE(75)
   END SUBROUTINE readin
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine  dateStampPrint
    integer    :: out_unit
    character(8)  :: date
    character(10) :: time
    character(5)  :: zone
    integer,dimension(8) :: values
    character( len = 9 ), parameter, dimension(12) :: month = (/ &
    'January  ', 'February ', 'March    ', 'April    ', &
    'May      ', 'June     ', 'July     ', 'August   ', &
    'September', 'October  ', 'November ', 'December ' /)

    ! call the interior function
    call date_and_time(date,time,zone,values)
    write(*,*) "################################################################"
    write(*,*) "         Demo code: MPI_PACK created by Wenqiang Feng"
    write (*, '(15x,a,a1,i2,a1,i4,2x,i2,a1,i2.2,a1,i2.2,a1)' ) &
       trim ( month( values(2))),'-',values(3),'-',values(1),values(5),&
          ':', values(6), ':', values(7), '.'
    write(*,*) "Copyright (c) 2015 WENQIANG FENG. All rights reserved."
```

```
    write(*,*) "###################################################################"
end subroutine  dateStampPrint

END MODULE io
```

11.8 inputdata.dat

```
&inputdata

a = 0.00D-00

b = 4.00D-00

n = 4/
```

11.9 Results

1. Run with 3 processors (2 workers)

```
Global x:   0.0000000000000000       0.5000000000000000        1.5000000000000000
        2.5000000000000000        3.5000000000000000        4.0000000000000000
!-------------------------------------------------!
I am master, I am sending the following data.
 a=   0.0000000000000000       b=   4.0000000000000000       n=           4
!-------------------------------------------------!
local u (before):   10.000000000000000        11.000000000000000
     12.000000000000000        13.000000000000000
local u (before):   20.000000000000000        21.000000000000000
     22.000000000000000        23.000000000000000
local u (after):   10.000000000000000        11.000000000000000
       12.000000000000000        21.000000000000000
local u (after):   12.000000000000000        21.000000000000000
      22.000000000000000        23.000000000000000
collected data   10.000000000000000        11.000000000000000
      12.000000000000000        21.000000000000000
      22.000000000000000        23.000000000000000
###################################################################
         Demo code: MPI_PACK created by Wenqiang Feng
              December-13-2015  22:22:36.
Copyright (c) 2015 WENQIANG FENG. All rights reserved.
###################################################################
>>main>> MR timing=    3.2496452331542969E-004  sec on            2  WRs
```

2. Run with 5 processors (4 workers)

```
Global x:   0.0000000000000000       0.5000000000000000        1.5000000000000000
      2.5000000000000000        3.5000000000000000        4.0000000000000000
!-------------------------------------------------!
I am master, I am sending the following data.
 a=   0.0000000000000000       b=   4.0000000000000000       n=           4
!-------------------------------------------------!
```

```
local u (before):   10.000000000000000        11.000000000000000
   12.000000000000000
local u (before):   30.000000000000000        31.000000000000000
      32.000000000000000
local u (after):   21.000000000000000        31.000000000000000
      41.000000000000000
local u (after):   10.000000000000000        11.000000000000000
     21.000000000000000
local u (before):   20.000000000000000        21.000000000000000
       22.000000000000000
local u (after):   11.000000000000000        21.000000000000000
        31.000000000000000
local u (before):   40.000000000000000        41.000000000000000
     42.000000000000000
local u (after):   31.000000000000000        41.000000000000000
       42.000000000000000
collected data   10.000000000000000        11.000000000000000
      21.000000000000000        31.000000000000000
      41.000000000000000        42.000000000000000
################################################################
         Demo code: MPI_PACK created by Wenqiang Feng
              December-13-2015  22:24:28.
Copyright (c) 2015 WENQIANG FENG. All rights reserved.
################################################################
>>main>> MR timing=    4.9114227294921875E-004  sec on            4  WRs
```

References

[1] V. Alexiades, *Numerical Methods for Conservation Laws*. http://www.math.utk.edu/~vasili/578/ASSIGNMENTS.html, 2015. 1, 12

[2] Anonymity, *Core*. http://www.cnet.com/news/intels-next-gen-quad-core-processors-tested/. 5

[3] ——, *Intel's CPU architecture*. http://www.bit-tech.net/hardware/cpus/2011/01/03/intel-sandy-bridge-review/1. 3, 5

[4] ——, *Multi-Core*. https://en.wikipedia.org/wiki/Multi-core_processor. 5

[5] ——, *Node*. https://en.wikipedia.org/wiki/Node_(computer_science). 5

[6] ——, *Threads*. http://techterms.com/definition/thread. 6

[7] ——, *Threads .vs. subroutines*. http://math.hws.edu/eck/cs124/javanotes6/c12/s1.html. 3, 6

[8] U. Center for High-Performance Computing, *Running a Job on HPC using PBS*. https://hpcc.usc.edu/support/documentation/running-a-job-on-the-hpcc-cluster-using-pbs/, 2015. 14

[9] Intel, *Intel's dual- and quad-core processors*. http://www.cnet.com/news/intels-next-gen-quad-core-processors-tested/. 3, 6

[10] U. Maui High Performance Computing Center, *Introduction to Parallel Programming*. http://phi.sinica.edu.tw/tyuan/old.pages/pcfarm.19991228/aspac/instruct/workshop/html/parallel-intro/ParallelIntro.html, 1995. 3, 7

[11] F. Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 3.0*, Message Passing Interface Forum, 2012. 7, 8

[12] F. Stack Exchange, *MPICH vs OpenMPI*. http://stackoverflow.com/questions/2427399/mpich-vs-openmpi, 2014. 8

[13] U. The National Institute for Computational Sciences, *Running Jobs*. https://www.nics.tennessee.edu/computing-resources/darter/running_jobs, 2015. 14

[14] Wikipedia, *SPMD*. https://en.wikipedia.org/wiki/SPMD, 2015. 7