松山湖材料实验室

中國科學院物理研究所
Institute of Physics, Chinese Academy of Sciences

材料计算与数据库平台

MPI Basics

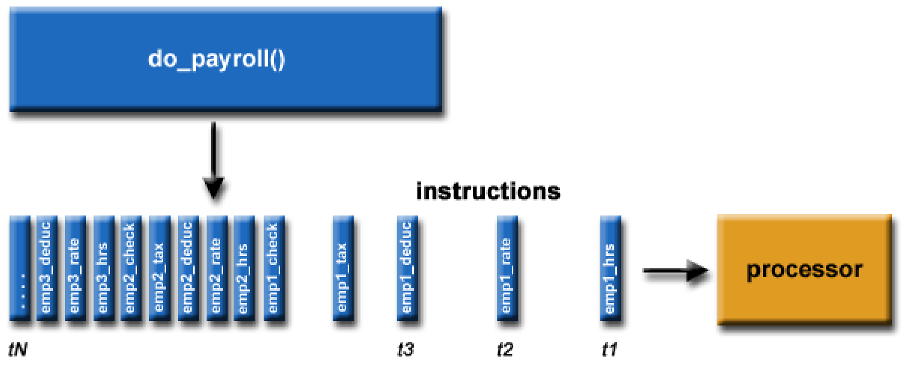
高恒

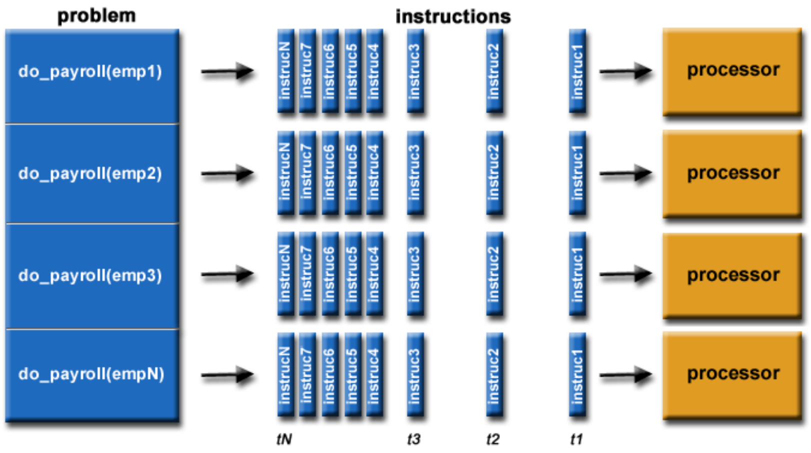
2020-2-28

Contents

Parallel computing

- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed

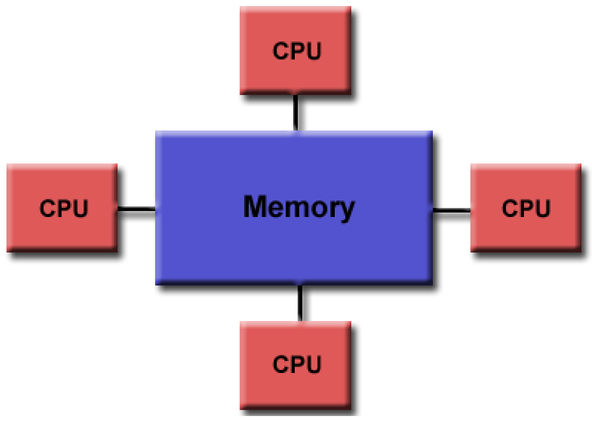




Two basic approaches

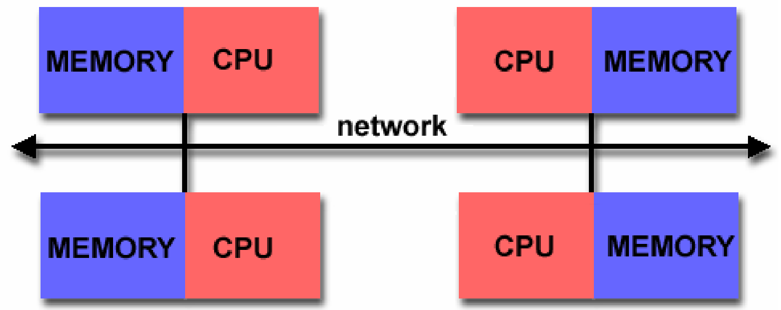
Shared Memory Computer

- Used by most laptops/PCs
- Multiple cores (CPUs)
- Share a global memory space
- Cores can efficiently exchange/share data



Distributed Memory

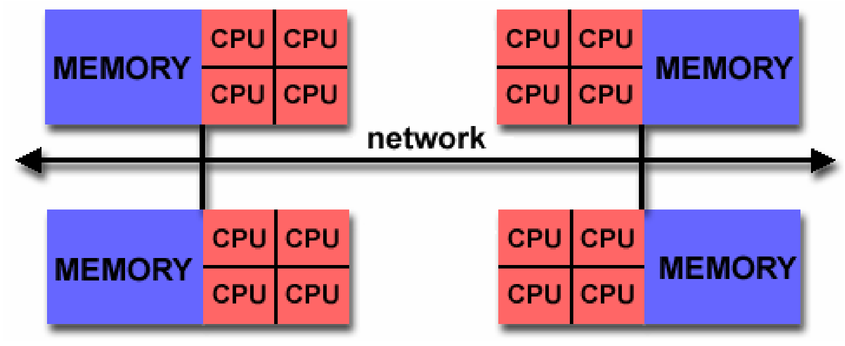
- Collection of nodes which have multiple cores
- Each node uses its own local memory
- Work together to solve a problem
- Communicate between nodes and cores via messages
- Nodes are networked together



Parallel programming models

Directive-based parallel programming language
- OpenMP (most widely used)
- High Performance Fortran (HPF) is another example
- Directives tell processor how to distribute data and work across the processors
- Directives appear as comments in the serial code
- Implemented on shared memory architectures

Message Passing
- MPI (most widely used)
- Pass messages to send/receive data between processes
- Each process has its own local variables
- Can be used on either shared or distributed memory architectures



Hybrid Distributed-Shared Memory

Hybrid MPI/openMP programming

MPI and OpenMP

MPI (Message Passing Interface)

- standardized library (not a language)
- collection of processes communicating via messages
  available for most architectures
- http://www.mpi-forum.org/

OpenMP

- API for shared memory programming
- available on most architectures as a compiler extension (C/C++, Fortran)
- includes compiler directives, library routines and environment variables
- www.openmp.org

Threads and Processes

• Process

have own address space
can have multiple threads

• MPI

many processes
shared-nothing architecture
explicit messaging
implicit synchronization
all or nothing parallelization

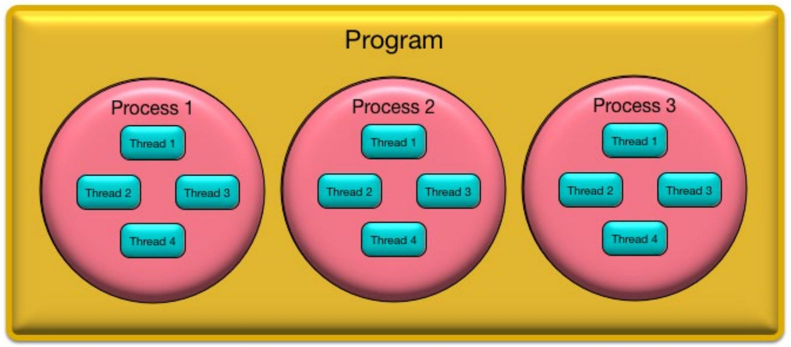
• Thread

execute within process
same address space
share process's stack
thread specific data

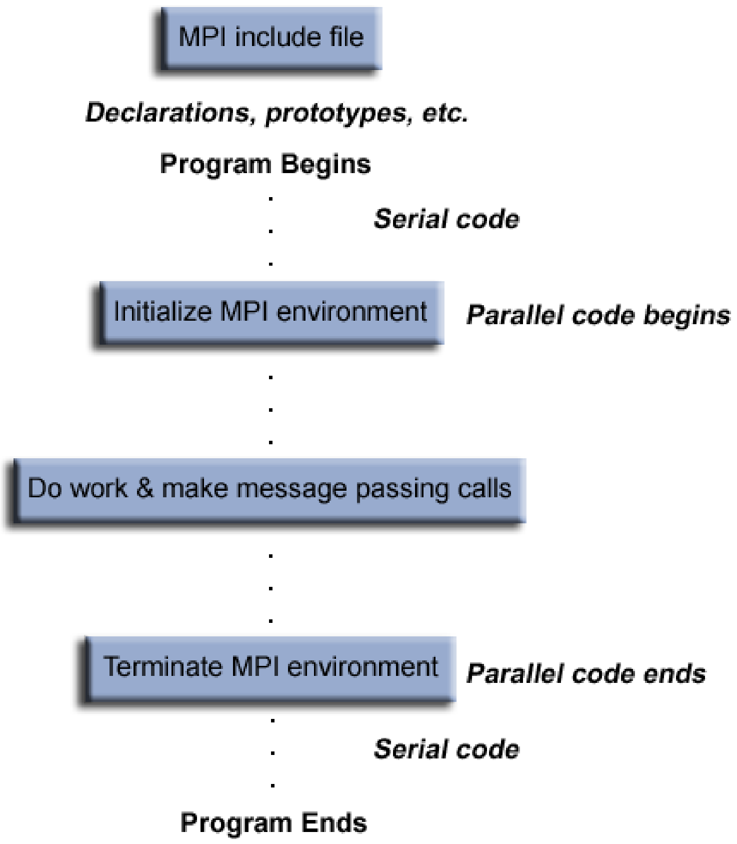
• OpenMP

1 process, many threads
shared-everything architecture
implicit messaging
explicit synchronization
incremental parallelism

Hybrid MPI/OpenMP program



MPI program


MPI include file
Declarations, prototypes, etc.
Program Begins
Serial code
Initialize MPI environment
Parallel code begins
Do work & make message passing calls
Terminate MPI environment
Parallel code ends
Serial code
Program Ends

```
#include "mpi.h"
#include <stdio.h>

int main(int argc, char *argv[]) {
int  numtasks, rank, len, rc;
char hostname[MPI_MAX_PROCESSOR_NAME];

rc = MPI_Init(&argc,&argv);
if (rc != MPI_SUCCESS) {
  printf ("Error starting MPI program. Terminating.\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  }

MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Get_processor_name(hostname, &len);
printf ("Number of tasks= %d My rank= %d
           Running on %s\n", numtasks,rank,hostname);

/*******  do some work *******/

MPI_Finalize();
}
```

MPI program

- Header file：

C include file	Fortran include file
#include "mpi.h"	include 'mpif.h'

Calling MPI：

C Binding	
Format:	rc = MPI_Xxxxx(parameter, ...)
Example:	rc = MPI_Bsend(&buf,count,type,dest,tag,comm)
Error code:	Returned as "rc". MPI_SUCCESS if successful

Fortran Binding	
Format:	CALL MPI_XXXXX(parameter,..., ierr)
	call mpi_xxxxx(parameter,..., ierr)
Example:	CALL MPI_BSEND(buf,count,type,dest,tag,comm,ierr)
Error code:	Returned as "ierr" parameter. MPI_SUCCESS if successful

Compile and run

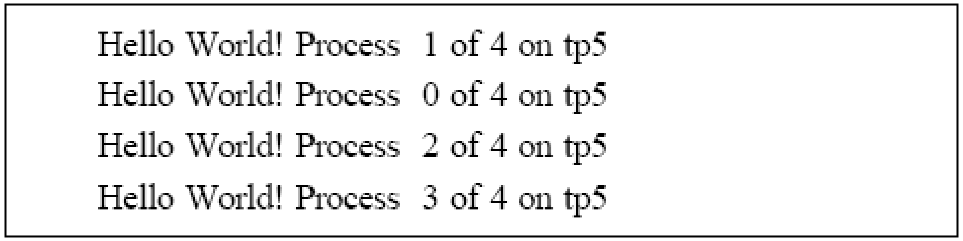
- Compile

C: mpicc -o mpiprog mpisrc.c

Fortran 77: mpif77 -o mpiprog mpisrc.f90

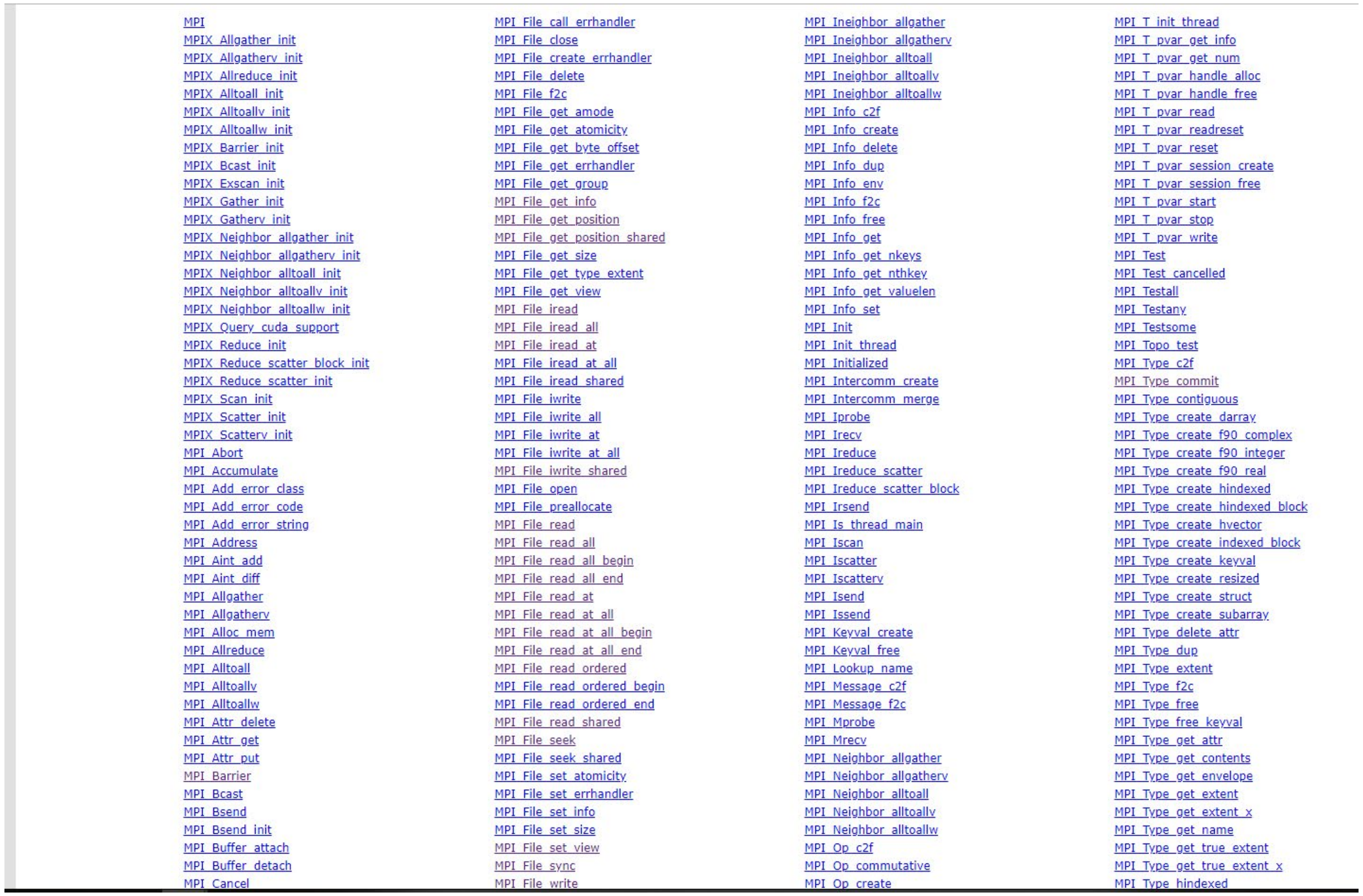
- Run

mpirun -np 4 ./mpiprog



```
Hello World! Process  1 of 4 on tp5
Hello World! Process  0 of 4 on tp5
Hello World! Process  2 of 4 on tp5
Hello World! Process  3 of 4 on tp5
```

MPI Library



- MPI
- MPIX_Allgather_init
- MPIX_Allgatherv_init
- MPIX_Allreduce_init
- MPIX_Alltoall_init
- MPIX_Alltoallv_init
- MPIX_Alltoallw_init
- MPIX_Barrier_init
- MPIX_Bcast_init
- MPIX_Exscan_init
- MPIX_Gather_init
- MPIX_Gatherv_init
- MPIX_Neighbor_allgather_init
- MPIX_Neighbor_allgatherv_init
- MPIX_Neighbor_alltoall_init
- MPIX_Neighbor_alltoallv_init
- MPIX_Neighbor_alltoallw_init
- MPIX_Query_cuda_support
- MPIX_Reduce_init
- MPIX_Reduce_scatter_block_init
- MPIX_Reduce_scatter_init
- MPIX_Scan_init
- MPIX_Scatter_init
- MPIX_Scatterv_init
- MPI_Abort
- MPI_Accumulate
- MPI_Add_error_class
- MPI_Add_error_code
- MPI_Add_error_string
- MPI_Address
- MPI_Aint_add
- MPI_Aint_diff
- MPI_Allgather
- MPI_Allgatherv
- MPI_Alloc_mem
- MPI_Allreduce
- MPI_Alltoall
- MPI_Alltoallv
- MPI_Alltoallw
- MPI_Attr_delete
- MPI_Attr_get
- MPI_Attr_put
- MPI_Barrier
- MPI_Bcast
- MPI_Bsend
- MPI_Bsend_init
- MPI_Buffer_attach
- MPI_Buffer_detach
- MPI_Cancel
- MPI_File_call_errhandler
- MPI_File_close
- MPI_File_create_errhandler
- MPI_File_delete
- MPI_File_f2c
- MPI_File_get_amode
- MPI_File_get_atomicity
- MPI_File_get_byte_offset
- MPI_File_get_errhandler
- MPI_File_get_group
- MPI_File_get_info
- MPI_File_get_position
- MPI_File_get_position_shared
- MPI_File_get_size
- MPI_File_get_type_extent
- MPI_File_get_view
- MPI_File_iread
- MPI_File_iread_all
- MPI_File_iread_at
- MPI_File_iread_at_all
- MPI_File_iread_shared
- MPI_File_iwrite
- MPI_File_iwrite_all
- MPI_File_iwrite_at
- MPI_File_iwrite_at_all
- MPI_File_iwrite_shared
- MPI_File_open
- MPI_File_preallocate
- MPI_File_read
- MPI_File_read_all
- MPI_File_read_all_begin
- MPI_File_read_all_end
- MPI_File_read_at
- MPI_File_read_at_all
- MPI_File_read_at_all_begin
- MPI_File_read_at_all_end
- MPI_File_read_ordered
- MPI_File_read_ordered_begin
- MPI_File_read_ordered_end
- MPI_File_read_shared
- MPI_File_seek
- MPI_File_seek_shared
- MPI_File_set_atomicity
- MPI_File_set_errhandler
- MPI_File_set_info
- MPI_File_set_size
- MPI_File_set_view
- MPI_File_sync
- MPI_File_write
- MPI_Ineighbor_allgather
- MPI_Ineighbor_allgatherv
- MPI_Ineighbor_alltoall
- MPI_Ineighbor_alltoallv
- MPI_Ineighbor_alltoallw
- MPI_Info_c2f
- MPI_Info_create
- MPI_Info_delete
- MPI_Info_dup
- MPI_Info_env
- MPI_Info_f2c
- MPI_Info_free
- MPI_Info_get
- MPI_Info_get_nkeys
- MPI_Info_get_nthkey
- MPI_Info_get_valuelen
- MPI_Info_set
- MPI_Init
- MPI_Init_thread
- MPI_Initialized
- MPI_Intercomm_create
- MPI_Intercomm_merge
- MPI_Iprobe
- MPI_Irecv
- MPI_Ireduce
- MPI_Ireduce_scatter
- MPI_Ireduce_scatter_block
- MPI_Irsend
- MPI_Is_thread_main
- MPI_Iscan
- MPI_Iscatter
- MPI_Iscatterv
- MPI_Isend
- MPI_Issend
- MPI_Keyval_create
- MPI_Keyval_free
- MPI_Lookup_name
- MPI_Message_c2f
- MPI_Message_f2c
- MPI_Mprobe
- MPI_Mrecv
- MPI_Neighbor_allgather
- MPI_Neighbor_allgatherv
- MPI_Neighbor_alltoall
- MPI_Neighbor_alltoallv
- MPI_Neighbor_alltoallw
- MPI_Op_c2f
- MPI_Op_commutative
- MPI_Op_create
- MPI_T_init_thread
- MPI_T_pvar_get_info
- MPI_T_pvar_get_num
- MPI_T_pvar_handle_alloc
- MPI_T_pvar_handle_free
- MPI_T_pvar_read
- MPI_T_pvar_readreset
- MPI_T_pvar_reset
- MPI_T_pvar_session_create
- MPI_T_pvar_session_free
- MPI_T_pvar_start
- MPI_T_pvar_stop
- MPI_T_pvar_write
- MPI_Test
- MPI_Test_cancelled
- MPI_Testall
- MPI_Testany
- MPI_Testsome
- MPI_Topo_test
- MPI_Type_c2f
- MPI_Type_commit
- MPI_Type_contiguous
- MPI_Type_create_darray
- MPI_Type_create_f90_complex
- MPI_Type_create_f90_integer
- MPI_Type_create_f90_real
- MPI_Type_create_hindexed
- MPI_Type_create_hindexed_block
- MPI_Type_create_hvector
- MPI_Type_create_indexed_block
- MPI_Type_create_keyval
- MPI_Type_create_resized
- MPI_Type_create_struct
- MPI_Type_create_subarray
- MPI_Type_delete_attr
- MPI_Type_dup
- MPI_Type_extent
- MPI_Type_f2c
- MPI_Type_free
- MPI_Type_free_keyval
- MPI_Type_get_attr
- MPI_Type_get_contents
- MPI_Type_get_envelope
- MPI_Type_get_extent
- MPI_Type_get_extent_x
- MPI_Type_get_name
- MPI_Type_get_true_extent
- MPI_Type_get_true_extent_x
- MPI_Type_hindexed

https://www.open-mpi.org/doc/current/

Basic Environment

MPI_Init(ierror)

- Initializes MPI environment
- Must be called in every MPI program
- Must be first MPI call
- Can be used to pass command line arguments to all

MPI_Finalize(ierror)

- Terminates MPI environment
- Last MPI function call

MPI_Comm_rank(comm, rank, ierror)

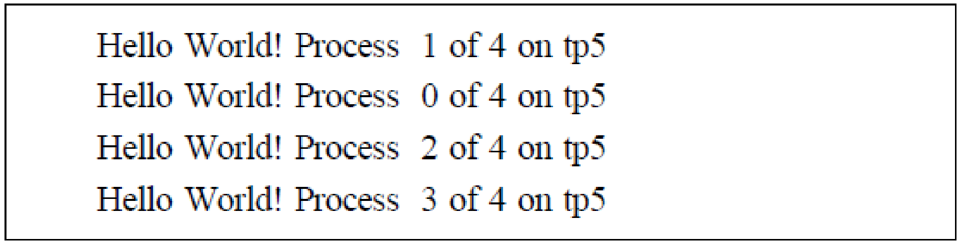
- Returns the rank of the calling MPI process
- Within the communicator, comm
- MPI_COMM_WORLD is set during Init(...)
- Other communicators can be created if needed

MPI_Comm_size(comm, size, ierror)

- Returns the total number of processes
- Within the communicator, comm

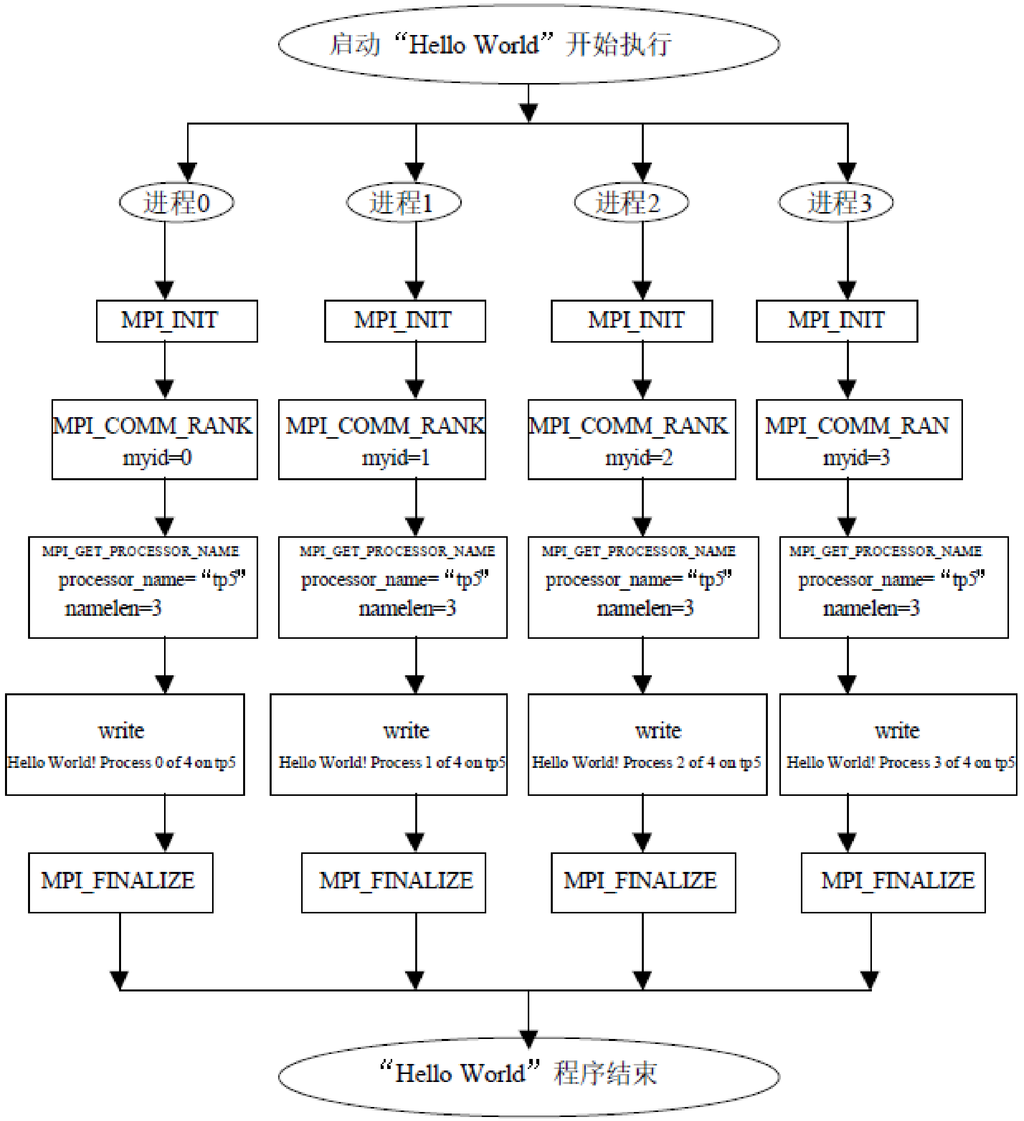
The first MPI program: “Hello World”

```
program main
include 'mpif.h'
character * (MPI_MAX_PROCESSOR_NAME) processor_name
integer myid, numprocs, namelen, ierr
call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
call MPI_GET_PROCESSOR_NAME(processor_name, namelen,
ierr)
write(*,10) myid,numprocs,processor_name
10 FORMAT('Hello World! Process ',I2,' of ',I1,' on ', 20A)
call MPI_FINALIZE(ierr)
end
```



```
Hello World! Process  1 of 4 on tp5
Hello World! Process  0 of 4 on tp5
Hello World! Process  2 of 4 on tp5
Hello World! Process  3 of 4 on tp5
```

The first MPI program: “Hello World”



启动 “Hello World” 开始执行
进程0
进程1
进程2
进程3
MPI_INIT
MPI_COMM_RANK myid=0
MPI_COMM_RANK myid=1
MPI_COMM_RANK myid=2
MPI_COMM_RAN myid=3
MPI_GET_PROCESSOR_NAME processor_name= “tp5” namelen=3
write Hello World! Process 0 of 4 on tp5
write Hello World! Process 1 of 4 on tp5
write Hello World! Process 2 of 4 on tp5
write Hello World! Process 3 of 4 on tp5
MPI_FINALIZE
“Hello World” 程序结束

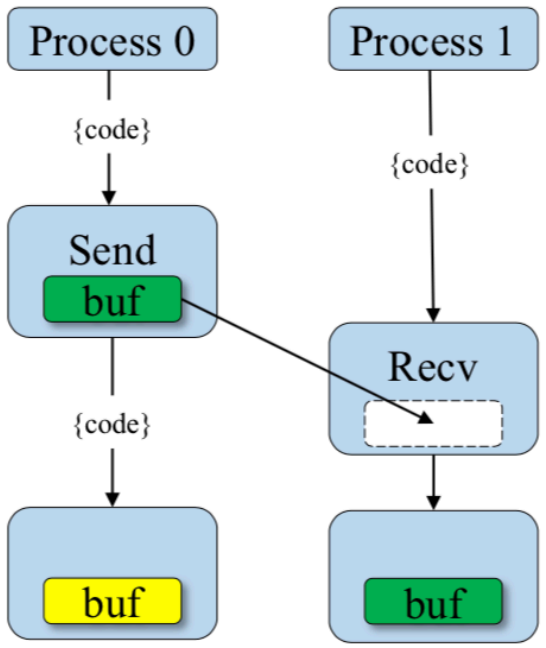
Point-to-point communication

MPI_Send(buf, count, datatype, dest, tag, comm, ierror)

- Send a message
- Returns only after buffer is free for reuse (Blocking)

MPI_Recv(buf, count, datatype, source, tag, comm, status, ierror)

- Receive a message
- Returns only when the data is available
- Blocking



MPI_Sendrecv(…)

- Two way communication
- Blocking

Data types

C Data Types		Fortran Data Types	
MPI_CHAR	signed char	MPI_CHARACTER	character
MPI_SHORT	signed short int		
MPI_INT	signed int	MPI_INTEGER	integer
MPI_LONG	signed long int		
MPI_UNSIGNED_CHAR	unsigned char		
MPI_UNSIGNED_SHORT	unsigned short int		
MPI_UNSIGNED	unsigned int		
MPI_UNSIGNED_LONG	unsigned long int		
MPI_FLOAT	float	MPI_REAL	real
MPI_DOUBLE	double	MPI_DOUBLE_PRECISION	double precision
MPI_LONG_DOUBLE	long double		
MPI_BYTE	8 binary digits	MPI_BYTE	8 binary digits
MPI_PACKED	MPI_Pack()	MPI_PACKED	MPI_Pack()

https://www.mpich.org/static/docs/latest/www3/Constants.html

MPI_Send and MPI_Recv: example

```fortran
program main
use mpi
integer, parameter :: ia=10
integer :: myid, nprocs, mpierr, mpistatus(mpi_status_size)
double precision :: a(ia)

call MPI_INIT(mpierr)
call MPI_COMM_RANK(mpi_comm_world, myid, mpierr)
call MPI_COMM_SIZE(mpi_comm_world, nprocs, mpierr)

a=0.0d0
if(myid==0) then
  do i=1, ia
    a(i)=i
  end do
end if

if(myid==0) then
  call MPI_SEND(a, ia, mpi_double_precision, 1, 101, mpi_comm_world, mpierr)
  call MPI_SEND(a(1), ia, mpi_double_precision, 2, 101, mpi_comm_world, mpierr)
  call MPI_SEND(a(1), ia/2, mpi_double_precision, 3, 101, mpi_comm_world, mpierr)
  call MPI_SEND(a(6), ia/2, mpi_double_precision, 3, 102, mpi_comm_world, mpierr)
end if
if(myid==1) then
  call MPI_RECV(a, ia, mpi_double_precision, 0, 101, mpi_comm_world, mpistatus, mpierr)
end if
if(myid==2) then
  call MPI_RECV(a(1), ia, mpi_double_precision, 0, 101, mpi_comm_world, mpistatus, mpierr)
end if
if(myid==3) then
  call MPI_RECV(a(6), ia/2, mpi_double_precision, 0, 101, mpi_comm_world, mpistatus, mpierr)
  call MPI_RECV(a(1), ia/2, mpi_double_precision, 0, 102, mpi_comm_world, mpistatus, mpierr)
end if

do i=0, nprocs-1
  call MPI_BARRIER(mpi_comm_world, mpierr)
  if(myid==i) then
    write(*,*) "processor:", myid
    write(*,"(10f8.2)") a
  end if
end do

call MPI_FINALIZE(mpierr)

end
```

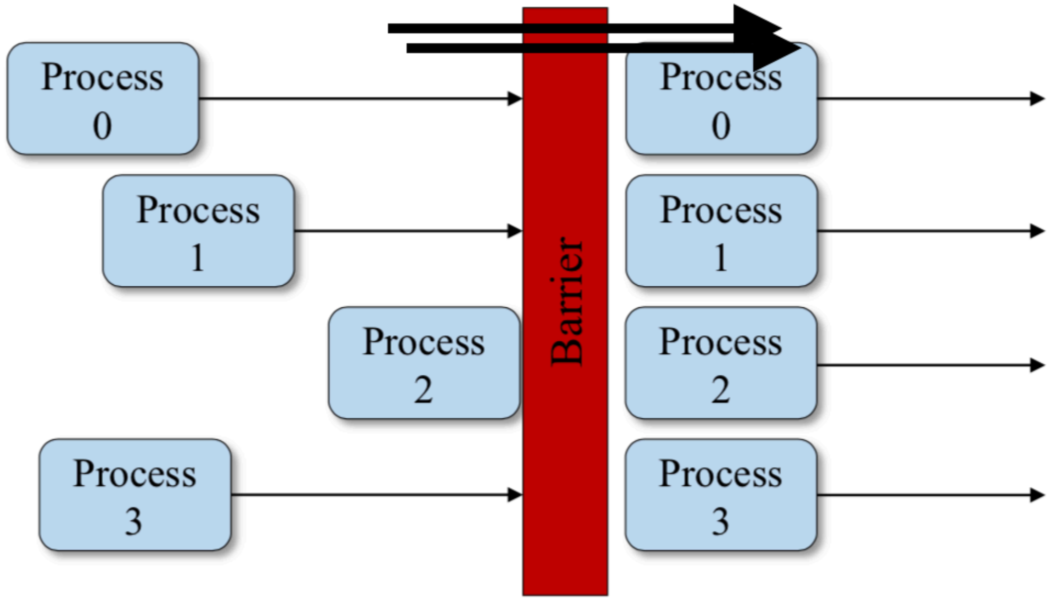
Collective communication

- MPI_Barrier
- **MPI_Bcast**
- **MPI_Gather/MPI_Gatherv**
- MPI_Allgather/MPI_Allgatherv
- **MPI_Scatter/MPI_Scatterv**
- MPI_Alltoall/MPI_Alltoallv
- **MPI_Reduce/MPI_Allreduce/MPI_Reduce_scatter**
- MPI_Scan
- …

Collective Communication (Barrier)

```
MPI_Barrier(comm, ierror)
```

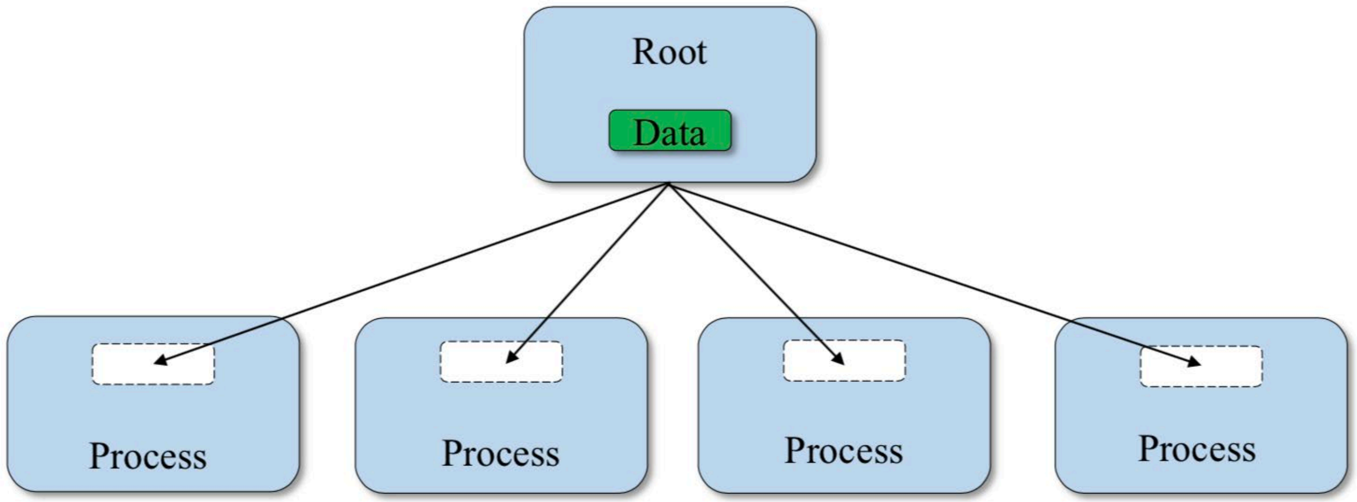
- Process synchronization (blocking)
- All processes forced to wait for each other
- Use only where necessary
- Will reduce parallelism



Collective Communication (Bcast)

MPI_Bcast(buffer, count, datatype, root, comm, ierror)

- Broadcasts a message from the root process to all other processes
- Useful when reading in input parameters from file

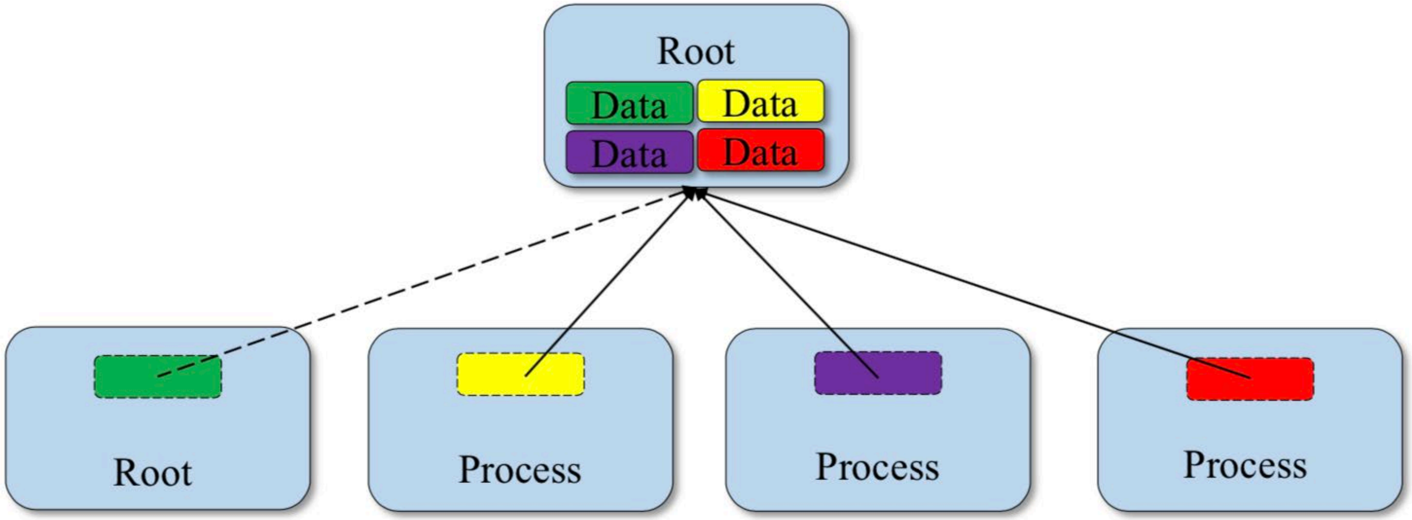


MPI_Bcast:example

```fortran
program main
use mpi
integer, parameter :: ia=10
integer :: myid, nprocs, mpierr, mpistatus(mpi_status_size)
double precision :: a(ia)

call MPI_INIT(mpierr)
call MPI_COMM_RANK(mpi_comm_world, myid, mpierr)
call MPI_COMM_SIZE(mpi_comm_world, nprocs, mpierr)

a=0.0d0
if(myid==0) then
  do i=1, ia
    a(i)=i
  end do
end if

call MPI_BCAST(a, ia, mpi_double_precision, 0, mpi_comm_world, mpierr)

do i=0, nprocs-1
  call MPI_BARRIER(mpi_comm_world, mpierr)
  if(myid==i) then
    write(*,*) "processor:", myid
    write(*,"(10f8.2)") a
  end if
end do

call MPI_FINALIZE(mpierr)

end
```

Collective Communication (Gather)

```
MPI_Gather(sendbuf, sendcount, sendtype, recvbuf,
           recvcount, recvtype, root, comm, ierror)
```



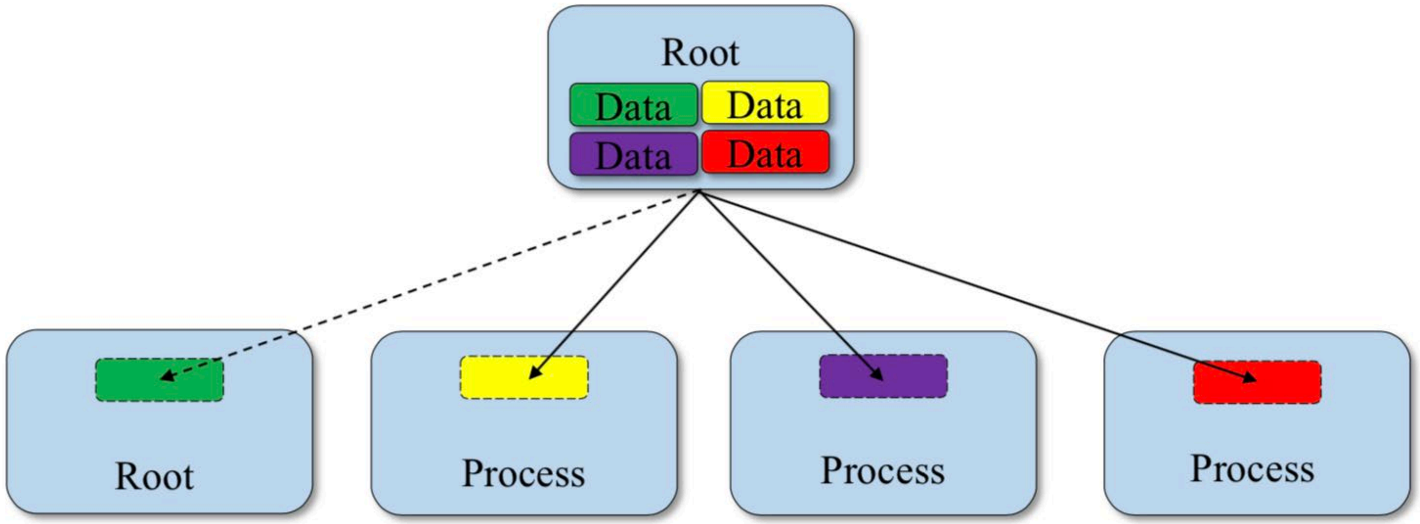
MPI_Gather: example

```fortran
program main
use mpi
integer, parameter :: ia=10
integer :: myid, nprocs, mpierr, mpistatus(mpi_status_size)
double precision :: a(ia)
double precision, allocatable :: b(:)

call MPI_INIT(mpierr)
call MPI_COMM_RANK(mpi_comm_world, myid, mpierr)
call MPI_COMM_SIZE(mpi_comm_world, nprocs, mpierr)

do i=1, ia
  a(i)=100d0*myid+i
end do
if(myid==0) then
  allocate(b(nprocs*ia))
  b=0.0d0
end if

call MPI_GATHER(a, ia, mpi_double_precision, b, ia, mpi_double_precision, 0, mpi_comm_world, mpierr)

do i=0, nprocs-1
  call MPI_BARRIER(mpi_comm_world, mpierr)
  if(myid==i) then
    write(*,*) "processor:", myid
    write(*,"(10f8.2)") a
  end if
end do

call MPI_BARRIER(mpi_comm_world, mpierr)
if(myid==0) then
  write(*,*) "b: (after gather)"
  write(*,"(40f8.2)") b
end if

call MPI_FINALIZE(mpierr)

end
```

Collective Communication (Scatter)

```
MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf,
            recvcount, recvtype,  root, comm, ierror)
```

- Sends individual messages from the root process to all other processes
- Opposite of Gather



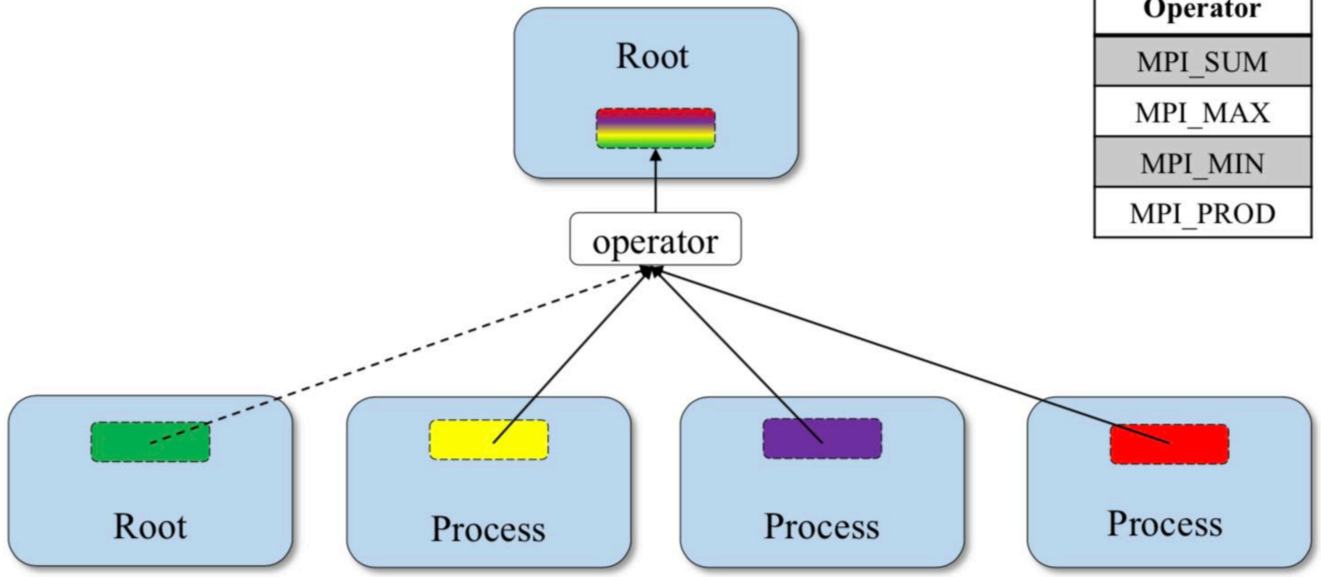
MPI_Scatter: example

```fortran
program main
use mpi
integer, parameter :: ia=10
integer :: myid, nprocs, mpierr, mpistatus(mpi_status_size)
double precision :: a(ia)
double precision, allocatable :: b(:)

call MPI_INIT(mpierr)
call MPI_COMM_RANK(mpi_comm_world, myid, mpierr)
call MPI_COMM_SIZE(mpi_comm_world, nprocs, mpierr)

do i=1, ia
  a(i)=100d0*myid+i
end do
if(myid==0) then
  allocate(b(nprocs*ia))
  b=0.0d0
end if

call MPI_GATHER(a, ia, mpi_double_precision, b, ia, mpi_double_precision, 0, mpi_comm_world, mpierr)

do i=0, nprocs-1
  call MPI_BARRIER(mpi_comm_world, mpierr)
  if(myid==i) then
    write(*,*) "processor:", myid
    write(*,"(10f8.2)") a
  end if
end do

call MPI_BARRIER(mpi_comm_world, mpierr)
if(myid==0) then
  write(*,*) "b: (after gather)"
  write(*,"(40f8.2)") b
end if

a=0.0d0
call MPI_SCATTER(b, ia/2, mpi_double_precision, a, ia/2, mpi_double_precision, 0, mpi_comm_world, mpierr)

do i=0, nprocs-1
  call MPI_BARRIER(mpi_comm_world, mpierr)
  if(myid==i) then
    write(*,*) "processor:", myid
    write(*,"(10f8.2)") a
  end if
end do
call MPI_FINALIZE(mpierr)
end
```

Collective Communication (reduce)

MPI_Reduce(sendbuf, recvbuf, count, datatype, op,
root, comm, ierror)

- Applies reduction operation on data from all processes
- Puts result on root process



Operator
MPI_SUM
MPI_MAX
MPI_MIN
MPI_PROD

Collective Communication (reduce)

MPI Reduction Operation		C Data Types	Fortran Data Type
MPI_MAX	maximum	integer, float	integer, real, complex
MPI_MIN	minimum	integer, float	integer, real, complex
MPI_SUM	**sum**	**integer, float**	**integer, real, complex**
MPI_PROD	product	integer, float	integer, real, complex
MPI_LAND	logical AND	integer	logical
MPI_BAND	bit-wise AND	integer, MPI_BYTE	integer, MPI_BYTE
MPI_LOR	logical OR	integer	logical
MPI_BOR	bit-wise OR	integer, MPI_BYTE	integer, MPI_BYTE
MPI_LXOR	logical XOR	integer	logical
MPI_BXOR	bit-wise XOR	integer, MPI_BYTE	integer, MPI_BYTE
MPI_MAXLOC	max value and location	float, double and long double	real, complex,double precision
MPI_MINLOC	min value and location	float, double and long double	real, complex, double precision

MPI_Reduce: example

```fortran
program main
use mpi
integer, parameter :: ia=10
integer :: myid, nprocs, mpierr, mpistatus(mpi_status_size)
integer :: iseed(4096)
double precision :: a(ia), b(ia), c(ia), d(ia), e(ia)

call MPI_INIT(mpierr)
call MPI_COMM_RANK(mpi_comm_world, myid, mpierr)
call MPI_COMM_SIZE(mpi_comm_world, nprocs, mpierr)

iseed=myid
call random_seed(put=iseed)
call random_number(a)
b=0.0d0
c=0.0d0

do i=0, nprocs-1
  call MPI_BARRIER(mpi_comm_world, mpierr)
  if(myid==i) then
    write(*,*) "processor:", myid
    write(*,"(10f8.3)") a
  end if
end do

call MPI_REDUCE(a, b, ia, mpi_double_precision, mpi_max, 0, mpi_comm_world, mpierr)
call MPI_REDUCE(a, c, ia, mpi_double_precision, mpi_min, 0, mpi_comm_world, mpierr)
call MPI_REDUCE(a, d, ia, mpi_double_precision, mpi_sum, 0, mpi_comm_world, mpierr)
call MPI_REDUCE(a, e, ia, mpi_double_precision, mpi_prod, 0, mpi_comm_world, mpierr)

call MPI_BARRIER(mpi_comm_world, mpierr)
if(myid==0) then
  write(*,*) "b: (max)"
  write(*,"(10f8.3)") b
  write(*,*) "c: (min)"
  write(*,"(10f8.3)") c
  write(*,*) "d: (sum)"
  write(*,"(10f8.3)") d
  write(*,*) "e: (prod)"
  write(*,"(10f8.3)") e
end if

call MPI_FINALIZE(mpierr)

end
```

Install mpi4py

安装依赖

•一个 MPI 实现软件，最好能支持 MPI-3 标准，并且最好是动态编译的。比较常用的 MPI 实现软件有 OpenMPI，MPICH 等。

•Python 2.7，Python 3.3+

wget https://repo.anaconda.com/archive/Anaconda3-2019.10-Linux-x86_64.sh

使用 pip 或者conda安装mpi4py

```
[gaoheng@login2 mpi]$ pip install mpi4py
```

pip

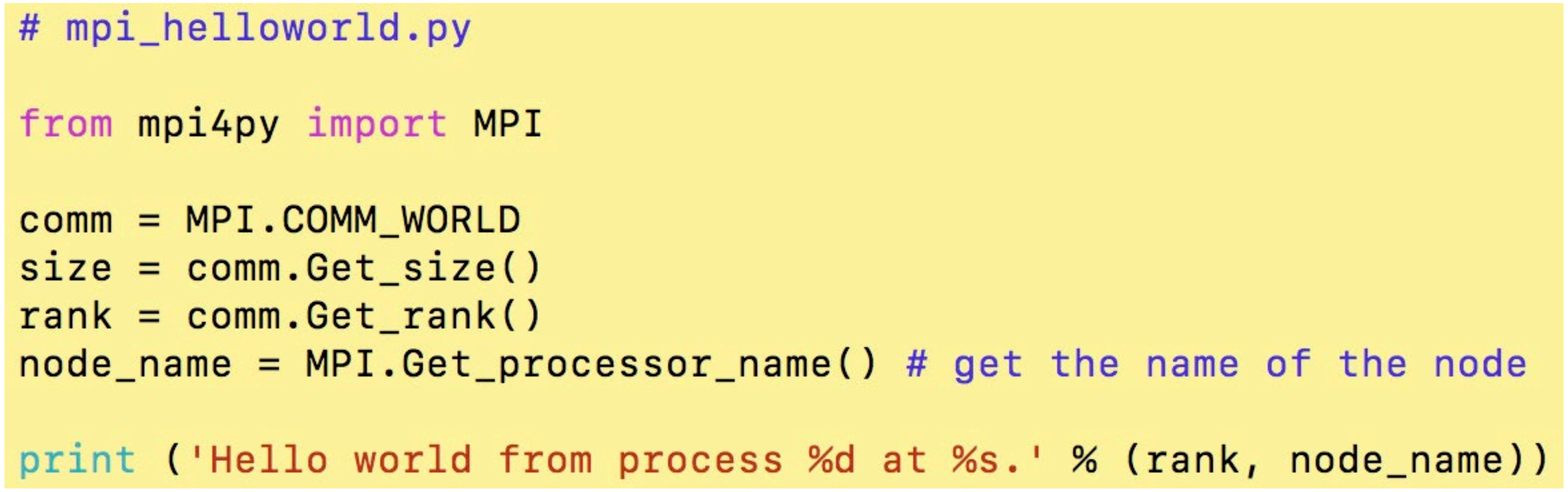
```
[gaoheng@login2 mpi]$ conda install mpi4py
```

conda

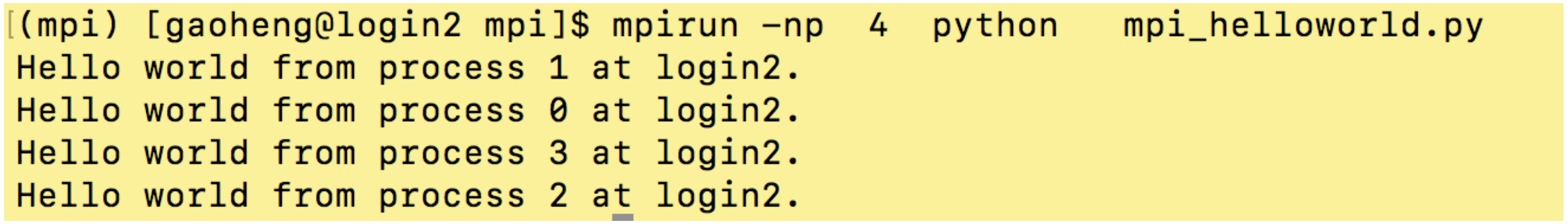
从源文件安装

从 https://pypi.org/project/mpi4py/ 上下载 mpi4py 安装包，然后解压安装包

Test mpi4py



```python
# mpi_helloworld.py

from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
node_name = MPI.Get_processor_name() # get the name of the node

print ('Hello world from process %d at %s.' % (rank, node_name))
```



```
[(mpi) [gaoheng@login2 mpi]$ mpirun -np  4  python   mpi_helloworld.py
Hello world from process 1 at login2.
Hello world from process 0 at login2.
Hello world from process 3 at login2.
Hello world from process 2 at login2.
```

mpi4py module

Communicators

MPI.Comm
MPI.Comm.Get_size()
MPI.Comm.Get_rank()
…

Point-to-Point Communications

MPI.Comm.Send()
MPI.Comm.Recv()
MPI.Comm.sendrecv()
…

Collective Communications

MPI.Comm.Bcast()
MPI.Comm.Scatter()
MPI.Comm.Reduce()
…

https://mpi4py.readthedocs.io/en/stable/index.html

A example: calculate π

$$\pi = \int_0^1 \frac{4}{1+x^2}dx \approx \frac{1}{n}\sum_{i=0}^{n-1} \frac{4}{1+(\frac{i+0.5}{n})^2}$$

Sequential version

```python
import math

def compute_pi(n):
    h = 1.0/n
    s = 0.0
    for i in range(n):
        x = h * (i + 0.5)
        s += 4.0 / (1.0 + x**2)
    return s*h

n = 1000
pi = compute_pi(n)
error = abs(pi - math.pi)
print ("pi is approximately %.10f, "  "error is %.10f" % (pi, error))
```

A example: calculate π

Parallel version

```python
from mpi4py import MPI
import math

def compute_pi(n, start=0, step=1):
    h = 1.0/n
    s = 0.0
    for i in range(start, n, step):
        x=h*(i+0.5)
        s+=4.0/(1.0+x**2)
    return s*h

comm = MPI.COMM_WORLD
nprocs = comm.Get_size()
myrank = comm.Get_rank()

if myrank == 0:
    n = 1000
else:
    n = None

n = comm.bcast(n, root=0)
mypi = compute_pi(n, myrank, nprocs)
pi = comm.reduce(mypi, op=MPI.SUM, root=0)

if myrank == 0:
    error = abs(pi - math.pi)
    print ("pi is approximately %.10f," "error is %.10f" % (pi, error))
```

Homeworks

Calculate π using Fortran or Python with MPI

Nilakantha Formula

$$\pi = 3 + \frac{4}{2\times 3\times 4} - \frac{4}{4\times 5\times 6} + \frac{4}{6\times 7\times 8} - \frac{4}{8\times 9\times 10} + \cdots$$

Viète's Formula

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2}\cdot\frac{\sqrt{2+\sqrt{2}}}{2}\cdot\frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2}\cdots.$$

Thanks for your attention！