Page :
Date : / /

Assignment No. -03

Aim :- Develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying intermediate sums calculated at different processors.

Objectives:- To learn about MPI, its benefits & its implementation.

Infrastructure :-

Software used :- Python, mpi4py library, Microsoft MPI v10.0 (https://www.microsoft.com/en-us/download/details.aspx?id=57467), Numpy.

Theory :-

With the advent of high-performance multicomputer, developers have been looking for message-oriented primitives that would allow them to easily write highly efficient applications. This means that primitives should be at a convenient level of abstraction & that their implementation incurs only minimal overhead.

Sockets were deemed insufficient for two reasons.

- First, they were at wrong level of abstraction by supporting only simple send & receive primitives.
- Second, sockets had been designed to communicate across networks using general-purpose protocol stacks such as TCP/IP. They were not considered suitable for the proprietary protocols developed for high-speed interconnection networks, such as those used in high-performance server clusters.

The result was that most interconnection networks & high-performance multicomputers were shipped with proprietary communication libraries. These libraries offered a wealth of high-level communication primitives. Of course, all libraries were mutually incompatible, so application developers now had a portability problem.

The need to be hardware & platform independent eventually led to definition of a standard for message passing, simply called the Message-Passing Interface or MPI, MPI is designed for parallel applications & as such is tailored to transient communication. It makes direct use of underlying network. Also, it assumes that serious failures such as process crashes or network partitions are fatal & do not require automatic recovery.

MPI assumes communication takes place within a known group of processes. Each group is assigned an identifier. Each process within a group is also assigned a (local) identifier. A (group ID, process ID) pair therefore uniquely identifies source or destination of message & is used instead of a transport-level address.

At the core of MPI are messaging primitives to support transient communication, of which the most intuitive ones are summarized.

Primitive	Meaning
MPI-bsend	Append outgoing message to a local send buffer
MPI-send	Send a message & wait until copied to local or remote buffer.
MPI-ssend	Send a message & wait until receipt starts
MPI-sendrecv	Send a message & wait for reply
MPI-isend	Pass reference to outgoing message, and continue

MPI-issend	Pass reference to outgoing message, & wait until receipt starts.
MPI-recv	Receive a message, block if there is none
MPI-irecv	Check if there is an incoming message, but do not block.

- **How MPI Works?**

Transient asynchronous communication is supported by means of MPI-bsend primitive. The sender submits a message for transmission, which is generally first copied to a local buffer in MPI runtime system. When message has been copied, the sender continues. The local MPI runtime system will remove message from its local buffer & take care of transmission as soon as a receiver has called a receive primitive.

There is also a blocking send operation, called MPI send, of which semantics are implementation dependent. The primitive MPIsend may either block the caller until specified message has been copied to MPI runtime system at sender's side, or until receiver has initiated a receive operation. Synchronous communication by which sender blocks until its request is accepted for further processing is available thro' MPI-ssend primitive. Finally, strongest form of synchronous comm. is also supported: When a sender calls MPIsendrecv, it sends a request to receiver & blocks until latter returns a reply. Basically, this primitive corresponds to a normal RPC.

Both MPIsend & MPIssend have variants that avoid copying messages from user buffers to buffers internal to the local MPI runtime system. These variants correspond to a form of asynchronous comm. With MPI-isend, a

sender passes a pointer to the message after
MPI runtime system takes care of communicati
sender immediately continues. To prevent overwr
message before comm. completes, MPI offers prim
to check for completion, or even to block if req
As with MPI_Send, whether message has actuall
transferred to receiver or that it has merely been
by local MPI runtime system to an internal buffe
unspecified. The operation MPI_Recv is called to
receive a message, it blocks caller until a mess
arrives. There is also an asynchronous variant,
MPI_Irecv, by which a receiver indicates that is p
to accept a message. The receiver can check whe
or not a message has indeed arrived, or block
one does.

- What is mpi4py?

MPI for Python provides MPI bindings for
language, allowing programmers to exploit m
processor computing systems. mpi4py is constru
on top of MPI-1/2 specifications & provides an
oriented interface which closely follows MPI-2

- Installing mpi4py :-

pip install mpi4py

- Microsoft MPI v10

Installing Process

1) Download the MPI exe file from the URL pro
2) Install the exe file.
3) Setup the path in the system variable to the

4) To verify the MPI software is installed correctly, open a new command prompt and type:
mpiexec -help
If everything is installed correctly, it will show the list of options available in the MPI.

- **Developing the code:-**

1) For distributed addition of the array of numbers we are going to use Reduce method available in the MPI operation.
2) For this first of all we are going to initialize the MPI, followed by the accepting the number of processes for performing the computation.
3) Then we assign rank to each process.
4) Initialize the array, and its value.
5) Send the sub-array of equal size to each process, where it gets computed.
6) This ~~command~~ computed value from each process is added together to get the global value or the total sum of the array.

- **Compiling the Code:-**

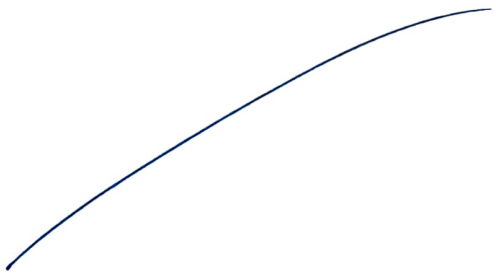
We run the code by the following command.
mpiexec -np 3 python sum.py
Here, 3 represent the number of processes.

- **Conclusion:-**

We learnt about MPI, how it was introduced and how to implement distributed computing with the help of MPI.

NAME: PALLAVI K. CHOPADE
ROLL NO.: 14
PRN NO.: 72036169K
BE (IT)
LP - V

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.rank

send_buf = []

if rank == 0:
    arr = np.array([12,21241,5131,1612251,161,6,161,1613,161363,12616,367,8363])
    arr.shape = (3, 4)
    send_buf = arr

v = comm.scatter(send_buf, root=0)
print("Local sum at rank {0}: {1}".format(comm.rank, np.sum(v)))

recvbuf = comm.reduce(v, root=0)
if comm.rank == 0:
    global_sum = np.sum(recvbuf)
    print("Global sum: "+ str(global_sum))
```



Assignment No.- 03

Name: - Pallavi Kamlakar Chopade.

Roll No.: - 14

PRN No.: - 72036169K

Subject: - Distributed Systems

Class: - BE(IT)