Ref. No.:

Date : / /

DS Assignment No.3

Aim: Develop a distributed system, to find sums of N elements in array by distributing N/n elements to n no of processors MPI or OpenMP. Demonstrate by displaying intermediate sum at diff position.

Obj: To learn about MPI, benefits & implementation

software used: Python, mpi4py library, Microsoft MPI V 10.0, Numpy

Theory:

With advent of high performance multicomputer developers have been looking for message oriented primitives that would allow to easily write highly efficient applications.

Sockets were deemed insufficient for 2 reasons:

- first they were at the wrong level of abstraction by supporting only send & receive primitive
- Second socket had been designed to communicate across networks protocol stacks such as TCP/IP.

They were not considered suitable for proprietary protocols developed for high speed interconnection networks, used in high performance clusters.

MPI assumes communication takes place within a known group of process. Each group is assigned

Ref. No.:
Date : / /

an identifier. A pair therefore uniquely identifies source or destination of message instead of transport level address. There may be several, possibly overlapping groups of processes involved in a computation & all are executing at same time.

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

Primitive	Meaning
MPI_bsend	Append outgoing msg to local send
MPI_send	Send message & wait until copied
MPI_ssend	Send message until receipt starts
MPI_sendrecv	Send message & wait for reply
MPI_isend	Pass ref to outgoing message
MPI_issend	Pass ref to outgoing messg till it
MPI_recv	Receive msg, block if none.
MPI_irecv	Check if there is incoming msg don't block.

How MPI works?

- Transient asynchronous communication is supported by means of MPI bsend primitive. The local runtime system will remove message from local buffer & call receive primitive.
There is also a blocking send operation called MPI send whose semantics are implementation dependent. Finally the st

Ref. No.:

Date : / /

form of synchronous communication is also supported. Whenever a sender calls MPI_sendrecv, it sends request to reciever and blocks until later returns a reply. Basically this primitive corresponds to normal RPC.

Both MPI_send mpi_ssend have variants that avoid copying messages from user buffers to local MPI runtime system. To prevent overwriting the message before communication completes, MPI offers primitives to check for completion. Or even block if required.

The operation MPI_recv is called to recieve a message; it blocks caller until message arrives. There is also synchronous variant, called MPI_irecv, by which reciever indicates that it is prepared to accept message. The reciever can check whether or not message has indeed arrived or block until one does.

What is mpi4py?

- MPI for Python provides MPI binding for python language, allowing programmers to exploit multiple processor computing system. mpi4py is constructed on top of MPI-1/2 specifications and provides OOP which closely follows MPI-2 C++ bindings

Ref. No.:

Date : / /

→ Installing mpi4py

pip install mpi4py

→ Installation process:

1) Download mpi exe file from URL provided above
2) Install exe file
3) Setup path in environment variable
4) Verify and type in cmd,
   mpiexec -help

→ Developing code:

1) For distributed addition, of array of nos we going to use Reduce method available in MPI operation.
2) For this first of all we are going to initialize the mpi followed by computation.
3) Assign rank to each process.
4) Initialize array & its value.
5) Send sub array of equal size to each process where it gets computed.
6) This computed value is added together to get sum of array.

→ Compiling the code:

mpiexec -np 3 python sum.py

Ref. No.: 3

Date : / /

Here 3 represent no of process

→ Conclusion : We learnt about MPI, how it was introduced & how to implement distributed computing with help of MPI.