Assignment No. 3

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 the intermediate sums calculated at different processors.

Objectives: To learn about MPI, its benefits, and its implementation.
Infrastructure:
Software used: Python, mpi4py library, Microsoft MPI v10.0 (https://www.microsoft.com/en-us/download/details.aspx?id=57467), Numpy

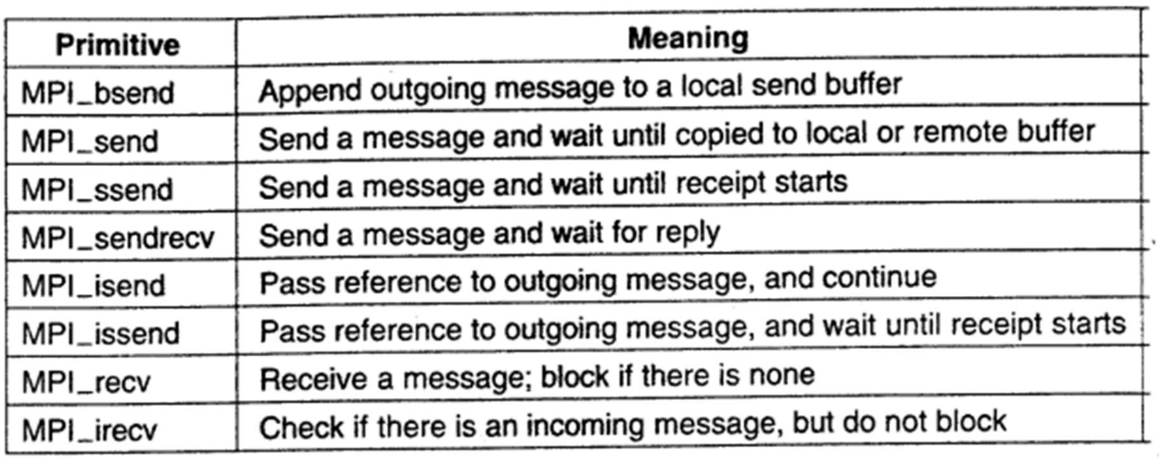
Theory:
With the advent of high-performance multicomputer, developers have been looking for message-oriented primitives that would allow them to easily write highly efficient applications. This means that the primitives should be at a convenient level of abstraction (to ease application development), and that their implementation incurs only minimal overhead.
Sockets were deemed insufficient for two reasons.

- First, they were at the wrong level of abstraction by supporting only simple send and receive primitives.
- Second, sockets had been designed to communicate across networks using general-purpose protocol stacks such as TCPIIP.

They were not considered suitable for the proprietary protocols developed for high-speed interconnection networks, such as those used in high-performance server clusters. Those protocols required an 'interface that could handle more advanced features, such as different forms of buffering and synchronization.
The result was that most interconnection networks and high-performance multicomputers were shipped with proprietary communication libraries. These libraries offered a wealth of high-level and generally efficient communication primitives. Of course, all libraries were mutually incompatible, so that application developers now had a portability problem.
The need to be hardware and platform independent eventually led to the definition of a standard for message passing, simply called the Message-Passing Interface or MPI. MPI is designed for parallel applications and as such is tailored to transient communication. It makes direct use of the underlying network. Also, it assumes that serious failures such as process crashes or network partitions are fatal and 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/D, process/D) pair therefore uniquely identifies the source or destination of a message, and is used instead of a transport-level address. There may be several, possibly overlapping groups of processes involved in a computation and that are all executing at the same time.

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 and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and 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

Figure 4-16. Some of the most intuitive message-passing primitives of MPI.

How MPI Works?

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

There is also a blocking send operation, called MPLsend, of which the sem antics are implementation dependent. The primitive MPLsend may either block the caller until the specified message has been copied to the MPI runtime system at the sender's side, or until the receiver has initiated a receive operation. Syn chronous communication by which the sender blocks until its request is accepted for further processing is available through the MPI~ssend primitive. Finally, the strongest form of synchronous communication is also supported: when a sender calls MPLsendrecv, it sends a request to the receiver and blocks until the latter returns a reply. Basically, this primitive corresponds to a normal RPC.

Both MPLsend and MPLssend have variants that avoid copying messages from user buffers to buffers internal to the local MPI runtime system. These vari ants correspond to a form of asynchronous communication. With MPI_isend, a sender passes a pointer to the message after which the MPI runtime system takes care of communication. The sender immediately continues. To prevent overwrit ing the message before communication completes, MPI offers primitives to check for completion, or even to block if required. As with MPLsend, whether the mes sage has actually been transferred to the receiver or that it has merely been copied by the local MPI runtime system to an internal buffer is left unspecified. Likewise, with MPLissend, a sender also passes only a pointer to the :MPI runtime system. When the runtime system indicates it has processed the message, the sender is then guaranteed that the receiver has accepted the message and is now working on it. The operation MPLrecv is called to receive a message; it blocks the caller until a message arrives. There is also an asynchronous variant, called MPLirecv, by which a receiver indicates that is prepared to accept a message. The receiver can check whether or not a message has indeed arrived, or block until one does.

What is mpi4py?

MPI for Python provides MPI bindings for the Python language, allowing programmers to exploit multiple processor computing systems. mpi4py is is constructed on top of the MPI-1/2 specifications and provides an object oriented interface which closely follows MPI-2 C++ bindings.

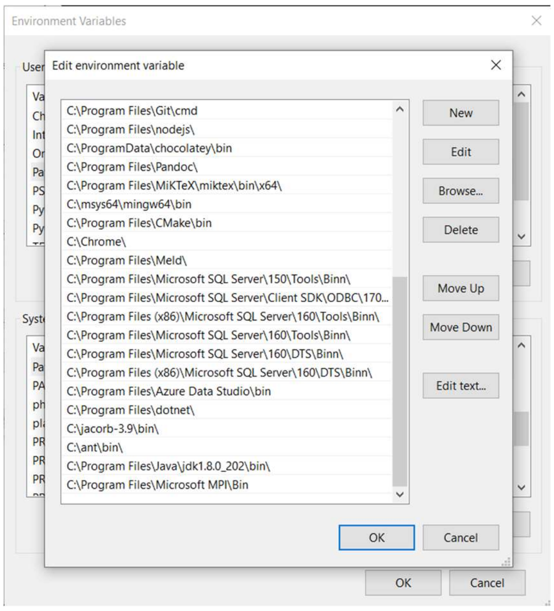
Installing mpi4py

```
pip install mpi4py
```

Microsoft MPI V10

Installation Process

1. Download the MPI exe file from the URL provided above.
2. Install the exe file.
3. Setup the path in the System Variable to the installed locations bin folder.



```
C:\Program Files\Git\cmd
C:\Program Files\nodejs\
C:\ProgramData\chocolatey\bin
C:\Program Files\Pandoc\
C:\Program Files\MiKTeX\miktex\bin\x64\
C:\msys64\mingw64\bin
C:\Program Files\CMake\bin
C:\Chrome\
C:\Program Files\Meld\
C:\Program Files\Microsoft SQL Server\150\Tools\Binn\
C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\170...
C:\Program Files (x86)\Microsoft SQL Server\160\Tools\Binn\
C:\Program Files\Microsoft SQL Server\160\Tools\Binn\
C:\Program Files\Microsoft SQL Server\160\DTS\Binn\
C:\Program Files (x86)\Microsoft SQL Server\160\DTS\Binn\
C:\Program Files\Azure Data Studio\bin
C:\Program Files\dotnet\
C:\jacorb-3.9\bin\
C:\ant\bin\
C:\Program Files\Java\jdk1.8.0_202\bin\
C:\Program Files\Microsoft MPI\Bin
```

i.e. As visible in the last line.

Note: This path may differ according to the path set at the installation time.

4. To verify the MPI software is installed correctly, open a new command prompt ant 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 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.