Basic MPI programs using point-ot-point library calls

Example 1.1

Write MPI Program to get started.

- **Objective**

  Write a MPI program to print "*Hello World*"

- **Description**

  Process with rank *k* (*k* = 1, 2, ..., *p*-1) will send"*Hello World*" message to process with rank*0*. Process with rank*0*receives the message and prints it. You have to use MPI *point-to-point* blocking communication library calls (MPI_Send and MPI_Recv) to write this program.

- **Input**

  None

- **Output**

  Process with rank *0* prints the following message with each of the other process's Rank Numbers (i.e. *rank* = 1,2, ..., p-1). Hello World from Process's *rank* where *rank* = *1,2, ..., p-1*

Example 1.2

Write MPI program (MPMD) to get started.

- **Objective**

  Write a MPI program to print "*Hello World*"

- **Description**

  Process with rank *k* (*k* = 1, 2, ..., *p*-1) will send " *Hello World*" message to process with rank *0*.Process with rank *0*receives the message and prints it. MPI *point-to-point* communication library calls (MPI_Send and MPI_Recv) have been used in this program. The master process is responsible for receiving the "hello message" from all participating processes (i.e slaves). On the other hand, the slave program is responsible for sending "Hello Message" to the master process. All slaves perform identical tasks. In other words, the slave program send the result back to the master process.

- **Input**

  None

- **Output**

Process with rank *0* prints the following message with each of the other process's Rank Numbers (i.e. *rank* = 1,2, ..., p-1). Hello World From process *rank* where *rank* = 1,2, ...p-1

Example 1.3

Write MPI program to find sum of n integers on Parallel Processing Platform. You have to use MPI point-to-point blocking communication library calls .

- **Objective**

  Write a MPI program to find sum of *n* integers on *p* processors of Message Passing cluster

- **Description**

  Each process with rank greater than *0* sends the value of its rank to the process with rank *0*. Process with rank *0*receives the values from each process and calculate the sum. Finally, process with rank *0* prints the sum.

- **Input**

  For input data, let each process uses its identifying number, i.e. the value of its rank. For example, process with rank*0* uses the number 0, process with rank *1* uses the number 1, etc.

- **Output**

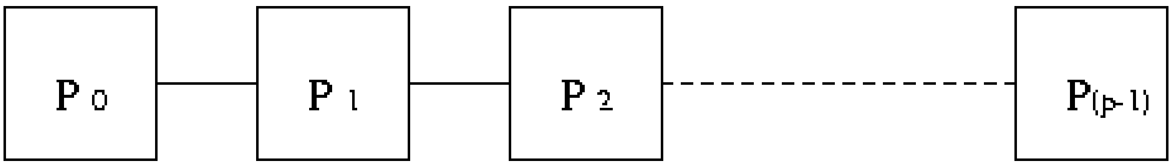
  Process with rank *0* prints the final sum.

Example 1.4

Write MPI program to find sum of n integers on Parallel Processing Platform. You have to use MPI point-to-point blocking communication library calls .

- **Description**

  In *linear array* interconnection network, each processor (except the processor at the end) has a direct communication link to the immediate next processor. A simple way of communicating a message between processors is, by repeatedly passing message to the processor immediately to either right or left , until it reaches its destination, i.e. last processor in the *linear array*. A simple way to connect processors is illustrated in Figure



All the processes are involved in communication. The processes with rank *k* (*k* is greater than *0*) receives the accumulated or partial sum from the previous process with rank *k*-1. Finally, process with rank *p*-1 prints the final sum.

- **Input**

  For input data,let each process use its identifying number, i.e. the value of its rank. For example, process with rank *0* uses the number 0, process with rank *1* uses the number 1, etc.

- **Output**

  Process with rank *p*-1 prints the final sum.

Example 1.5

Write MPI program to find sum of n integers on a Parallel Computing System in which processors are connected with ring topology and use MPI point-to-point blocking communication library calls .

- **Objective**

  Write a MPI program to find sum of *n* values using *p* processors of cluster. Assume that *p* processors are arranged in *ring topology*.

- **Description**

  In linear array interconnection network with a wraparound connection is called as a *ring*. A wraparound connection is often provided between the processors at the end. A simple way of communicating a message between processors is, by repeatedly passing message to the processor immediately to either right or left; depending on which direction yield a shorter path, until it reaches its destination, i.e., first processor in the *ring*.

  All the processes are involved in communication. The process with rank *k* (*k* is greater than *0*) receives the accumulated or partial sum from the previous process with rank *k*-1. Process with rank *p*-1 sends the final sum to process with rank *0*. Finally, process with rank *0* prints the final sum.

- **Input**

  For input data, let each process use its identifying number, i.e. the value of its rank. For example, process with rank *0*uses the number 0, process with rank *1* uses the number 1, etc.

- **Output**

  Process with rank *0* prints the final sum.

Example 1.6

Write MPI program to find sum of n integers on a Parallel Computing System in which processors are connected with tree topology (Associative-fan-in rule for tree can be assumed) and use MPI point-to-point blocking communication library calls.

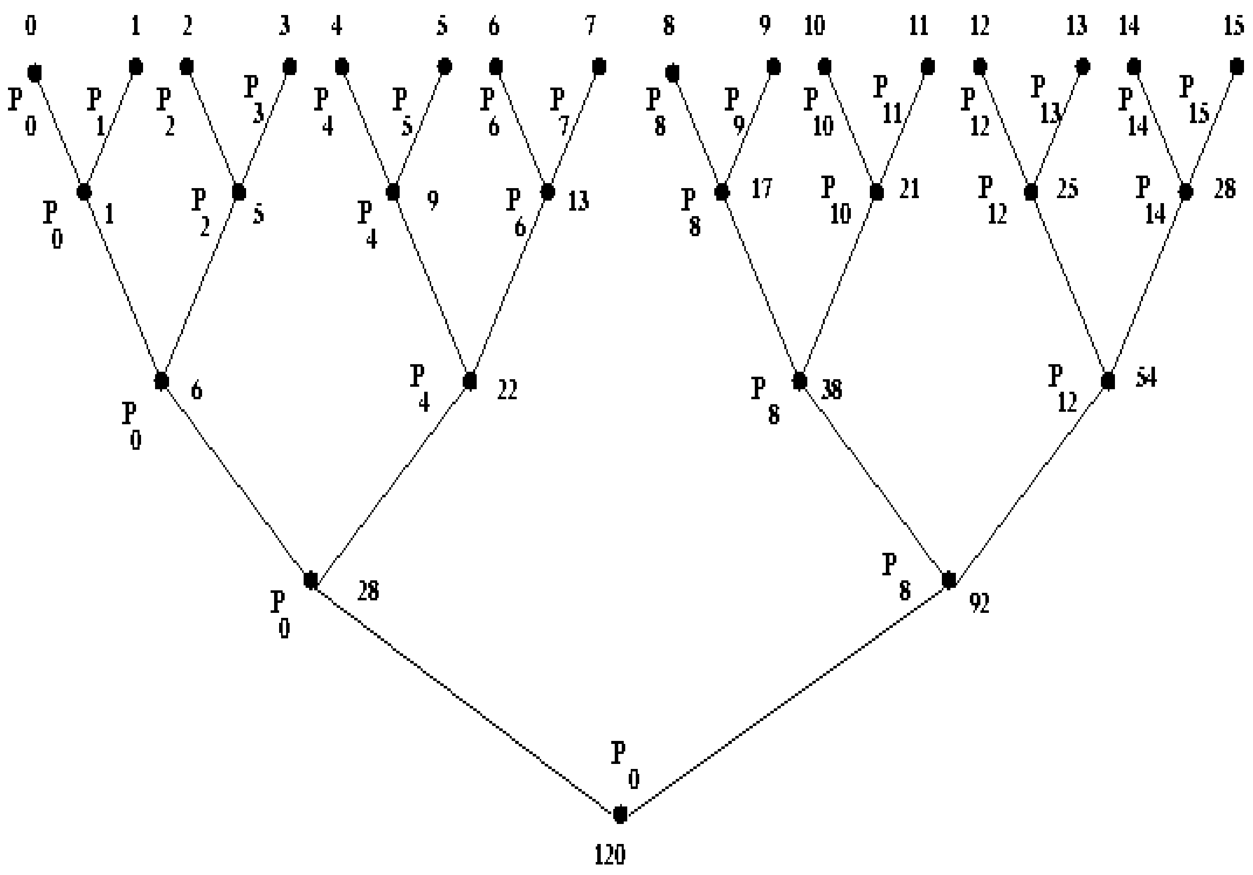
- **Objective**

  Write a MPI program to find sum of *n* (*n* = 2 *i* ; *i* = 3) integers on *p* (*p*=*n*) processors of cluster using *associative fan-in* rule.

- **Description**

  The *first step* is to group the total number of processors in ordered pairs such as (*P*0*, P*1), (*P*2*, p*3), (*P*4*, P*5), (*P*6*, p*7), ..., (*Pp-2, Pp-1*). Then compute the partial sums in all pairs of processors using MPI point-to-point blocking communication and accumulate the partial sum on the processors P0, P2, ...,P*p*-2. For example, the processor *Pi* (*i* is even) computes the partial sum for the pair (*Pi, Pi+*1) by performing MPI point-to-point blocking communication library calls.

  In the *second step*, consider the pair of processors (*P*0, *P*2), (*P*4, *P*6), ..., (*Pp*-4, *Pp*-2) obtain the new partial sum by considering the existing accumulated partial sums on the processors P0, P2, P4, ..., P8as explained in the previous step. This procedure is repeated till two processors are left out and finally accumulate the global sum on the processor *P*0.In this example, MPI point-to-point blocking communication library calls such as,**MPI_Send** and **MPI_Recv**, are used. An example of *associative fan-in* rule is described as a tree structure in the Figure 2 for *n*= 16



- **Input**

  For input data, let each process use its identifying number, i.e. the value of its rank. For example, process with rank *0*uses the number 0, process with rank *1* uses the number 1, etc.

- **Output**

  process with rank *0* prints the final sum.