DEI-FCTUC

Computação de Alto Desempenho – 2012/2013
Exercises #4: Introduction to the Message Passing Interface

Some of these exercises were taken or inspired from several sources including the Web.

Point-to-Point Communication

1. Using the MPI communication primitives, write a program in which a process sends a "Hello World" message to another process.

2. Consider the following code:

```
MPI_Comm_rank( MPI_COMM_WORLD, &myrank);
if (myrank == 0)    {
   MPI_Isend(buf1, 5, MPI_FLOAT, 1, 0, MPI_COMM_WORLD, req1);
   MPI_Isend(buf2, 5, MPI_FLOAT, 1, 0, MPI_COMM_WORLD, req2);
} else  {
   MPI_Irecv(buf1, 5, MPI_FLOAT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, req1);
   MPI_Irecv(buf2, 5, MPI_FLOAT, 0, 0, MPI_COMM_WORLD, req2);
}
```

Tell the correspondence between the operations done in this program by both processes. Is there any risk of dead-lock?

3. Now, do the same for the following code:

```
MPI_Comm_rank( MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
   MPI_Ssend(buf1, 10, MPI_FLOAT, 1, tag2, MPI_COMM_WORLD);
   MPI_Send(buf2, 10, MPI_FLOAT, 1, tag1, MPI_COMM_WORLD);
} else  {
   MPI_Irecv(buf1, 10, MPI_FLOAT, 0, tag1, MPI_COMM_WORLD, req1);
   MPI_Recv(buf2, 10, MPI_FLOAT, 0, tag2, MPI_COMM_WORLD, &st);
   MPI_Wait(req1, &st);
}
```

4. Consider the following code, where we could have a starvation problem:

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank( MPI_COMM_WORLD, &myrank);
if (myrank > 0)    { /*       PRODUTORES      */
   while (1) {
      MPI_Isend(b, n, MPI_FLOAT, 0, tag, MPI_COMM_WORLD, req);
      MPI_Wait(req, &status);
   }
} else  {              /*       CONSUMIDOR       */
   for (i = 1; i < size; i++)
      MPI_Irecv(&a[i-1], n, MPI_FLOAT, i, tag, MPI_COMM_WORLD, &reqT[i-1]);
   while (1) {
      MPI_Waitany(size-1, reqT, &index, &status);
      /*   tratar pedido correspondente a index  */
      MPI_Irecv(&a[index], n, MPI_FLOAT, index+1, tag,
                                  MPI_COMM_WORLD, &reqT[index]);
   }
}
```

How could you solve this problem?

Data Types

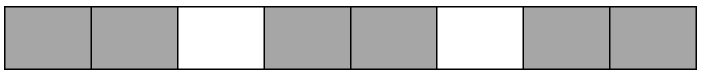
5. In the following exercises consider that you have a data structure made of 8 elements of type `char` and one `double`:

   ```
   struct base {
     double num;
     char character[8];
   };
   ```

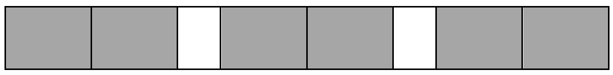
   If you have an array of 4 elements of type `struct base`, how could you create a new type for this array in MPI?

6. And what would happen if instead of an array of 4 consecutive elements you had the following patterns in memory?

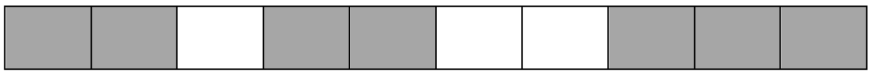
   a)



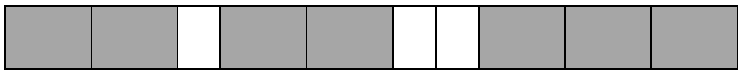
   b)



   c)

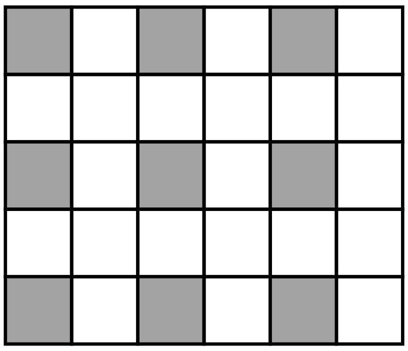


   d)



7. Consider the declaration of the following matrices:

   ```
   float a[5][6], e[3][3];
   ```



   Memory positions

   The figure above represents the matrix `a`. In this exercise you should extract the gray squares and store them in matrix e. You should use the functions `MPI_Type_vector()`, `MPI_Type_hvector()` and `MPI_Sendrecv()`.

Collective Communication

8. In this exercise we want to find the largest number inside an array. Consider that the coordinator of the group of processes reads the array from a file. To speed up the search, the coordinator distributes the array by the processes, which then send the result back. The coordinator should also participate in the search. The final result should be the position of the largest number in the array. You should use collective communication in this exercise. More precisely, you should use a scatter and a reduce (assume that the number of elements in the array is divisible by the number of processes).

9. In this exercise you should solve the N-body (many-body) problem. You will be given a sequential source code in C, and you should parallelize the code. You can use whatever parallelization algorithm you are pleased. You should use collective MPI communication. To help you with this task, you will be given a tiny program in Python, to generate random body masses and positions. In this exercise you should make some measurements to calculate the computation to communication ratio.