# Introdução ao MPI

**António Pina**
(pina@di.uminho.pt)

## Material de Apoio

- http://gec.di.uminho.pt/Minf/cpd1415/pcp/sumarios.html
- https://htor.inf.ethz.ch/teaching/mpi_tutorials/ppopp13/2013-02-24-ppopp-mpi-basic.pdf
- http://htor.inf.ethz.ch/teaching/mpi_tutorials/ppopp13/2013-02-24-ppopp-mpi-advanced.pdf
- http://mpitutorial.com/tutorials/mpi-hello-world
- http://mpitutorial.com/tutorials/mpi-send-and-receive/
- https://computing.llnl.gov/tutorials/mpi/

## Ambiente de trabalho

- SeARCH
- Máquina local com o openmpi instalado
- http://mpitutorial.com/tutorials/running-an-mpi-cluster-within-a-lan/
- https://www.open-mpi.org/video/internals/Cisco_JeffSquyres-1up.pdf

## Componente prática

1. Instalar o *openmpi* na máquina local ou, em alternativa, usar o s*earch*
2. Ligar usando as credenciais
   - search.di.uminho.pt

```
$ ssh userName@search.di.uminho.pt
      Password:???????
      Last login: Wed Sep 26 19:41:12 2018 from  ….
Rocks7.0(Manzanita)
Profile built 12:43 16-Apr-2019
Kickstarted 14:52 16-Apr-2019
[amp@search7 ~]$
```

3. O código abaixo ilustra um conjunto básico de primitivas disponíveis na API do MPI: ver (http://mpitutorial.com/tutorials/mpi-hello-world/ )

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
MPI_Init(&argc,&argv);  // Initialize the MPI environment
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size); // Get the number of processes

 int world_rank;
 MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);  // Get the rank of the process

 char processor_name[MPI_MAX_PROCESSOR_NAME];
 int name_len;
 MPI_Get_processor_name(processor_name, &name_len); // Get the name of the processor

 printf("Hello world from processor %s, rank %d out of %d processors\n",
     processor_name, world_rank, world_size);
 MPI_Finalize();  // Finalize the MPI environment. No more MPI calls can be made after this
}
```

4. Compilar
   - Local

```
amp@local$ make
   mpicc -o mpi_hello_world mpi_hello_world.c
```

   - Search

```
[amp@search7 MPI]$ make
   mpicc -o mpi_hello_world mpi_hello_world.c
```

5. Executar
   - Local

```
amp@local$ mpirun -n 2 --mca btl tcp,self    ./mpi_hello_world
   Hello world from processor redefixa.local, rank 1 out of 2 processors
   Hello world from processor redefixa.local, rank 2 out of 2 processors
```

Search

```
[amp@search7 MPI]$ cat mpi_hello_world.sl
#!/bin/bash
#SBATCH --job-name=mpi_hello_world

#numero de processos passado como parâmetro $1
mpirun -oversubscribe -n $1   --mca btl_tcp_if_exclude docker0 ./mpi_hello_world

[amp@search7 MPI]$ sbatch -N2 mpi_hello_world.sl 4
 Submitted batch job 10069
[amp@search7 MPI]$ squeue
  JOBID PARTITION  NAME    USER ST  TIME  NODES NODELIST(REASON)
  10069 cdados     bash    amp  R   0:04      2 compute-641-[10-11]
[amp@search7 MPI]$ cat slurm-1069.out

Hello world from processor compute-641-10.local, rank 0 out of 4 processors
Hello world from processor compute-641-10.local, rank 2 out of 4 processors
Hello world from processor compute-641-10.local, rank 1 out of 4 processors
Hello world from processor compute-641-10.local, rank 3 out of 4 processors
```

## Exercícios MPI

O código C que segue será usado como base para os exercícios propostos a seguir.

```
#include <mpi.h>
#include <stdio.h>
int main( int argc, char *argv[]) {
        int rank, msg;
        MPI_Status status;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank( MPI_COMM_WORLD, &rank );
        /* Process 0 sends and Process 1 receives */
        if (rank == 0) {
                msg = 123456;
                MPI_Send( &msg, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        }
        else if (rank == 1) {
                MPI_Recv( &msg, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
                printf( "Received %d\n", msg);
        }
        MPI_Finalize();
        return 0;
}
```

1) Altere o programa, acima, de forma o permitir o encadeamento de mensagens em anel considerando que o número de processos de arranque é **N**:
   a) O processo zero (rank=0) envia uma mensagem ao processo seguinte (*rank*=1) que pelo seu lado a reencaminha para o processo seguinte e assim sucessivamente.
   b) O mesmo que a alínea *a)* mas deve usar a primitiva **MPI_Comm_size** para conhecer o número total de processos.

2) O programa de base deverá ser alterado para permitir que o processo (*rank*=0) receba uma mensagem de cada um dos demais processos com um valor que identifica cada um dos respetivos *ranks*. No final a soma de todos os *ranks* é calculada pelo *rank*=0.

3) Altere o programa de forma a que um dos processos difunda uma mensagem para todos os processos. Seguidamente, todos os processos deverão chamar usar uma primitiva de redução, por exemplo **SUM**, que produzirá o somatório dos **ranks** de todos os processos na aplicação.

Exercício Complementar

4) Geração de Histogramas
   Enunciado
   histogram.c