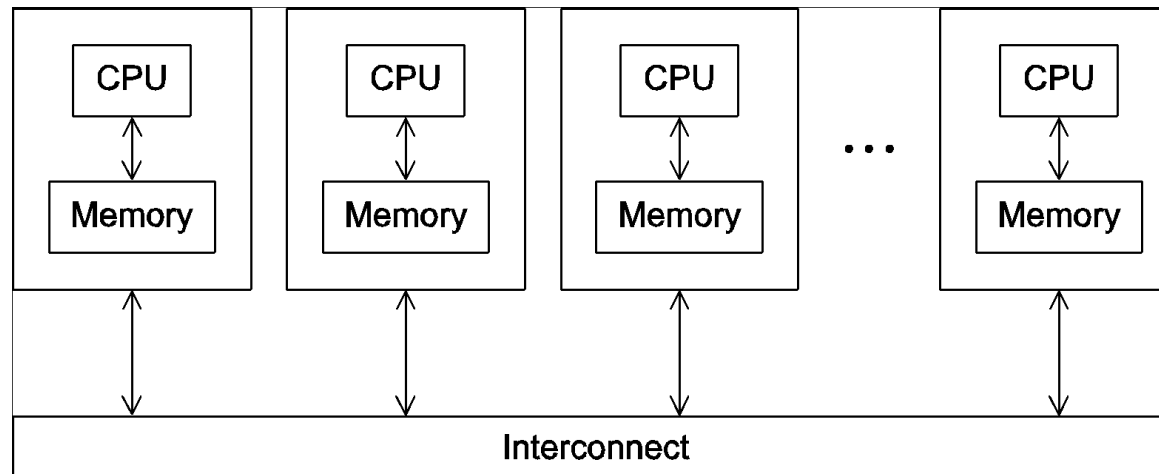


IPPD

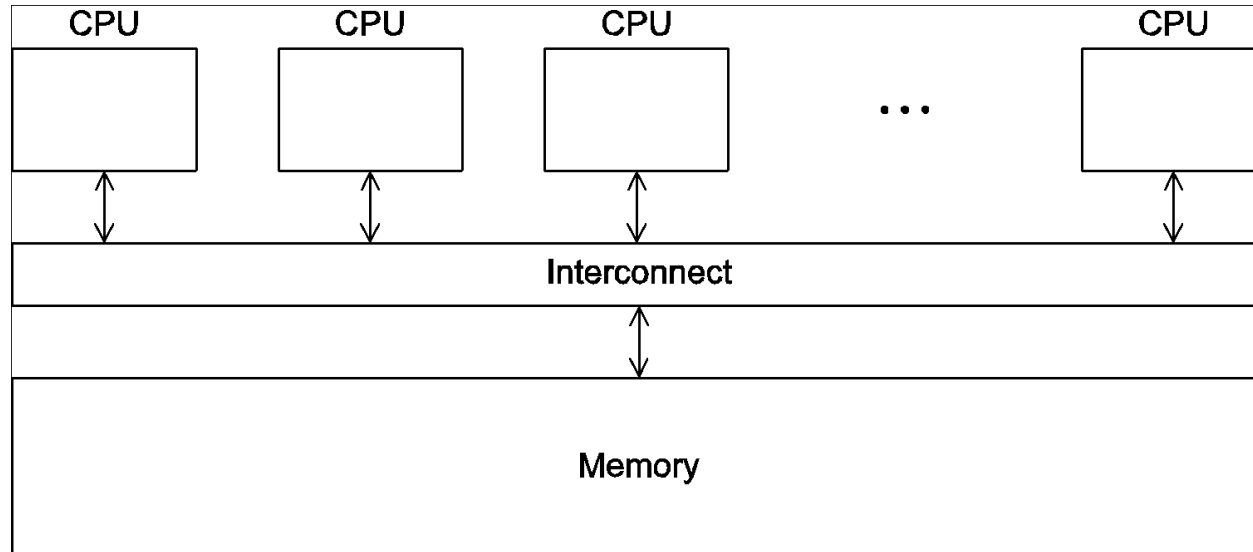
Hoje: Memória Distribuída

Prof. Dr. Rafael P. Torchelsen
rafael.torchelsen@inf.ufpel.edu.br

Memória Distribuída



Memória Compartilhada



Hello World!

```
#include <stdio.h>
```

```
int main(void) {  
    printf("hello, world\n");  
  
    return 0;  
}
```

Message Passing Interface (MPI) é um padrão para [comunicação de dados](#) em [computação paralela](#). Existem várias modalidades de computação paralela, e dependendo do problema que se está tentando resolver, pode ser necessário passar informações entre os vários [processadores](#) ou [nodos](#) de um [cluster](#), e o MPI oferece uma infraestrutura para essa tarefa.

No padrão MPI, uma aplicação é constituída por uma ou mais tarefas (as quais podem ser processos, ou threads, dependendo da implementação) que se comunicam, acionando-se funções para o envio e recebimento de mensagens entre os processos. Inicialmente, na maioria das implementações, um conjunto fixo de processos é criado. Porém, esses processos podem executar diferentes programas. Por isso, o padrão MPI é algumas vezes referido como **MPMD** (*multiple program multiple data*).

Implementações de MPI

- MPICH
 - <http://www.mpich.org/>
 - <http://www.mpich.org/documentation/guides/>
- Open MPI
 - <https://www.open-mpi.org>
 - <https://www.open-mpi.org/doc/>
- Microsoft MPI
 - [https://msdn.microsoft.com/en-us/library/bb524831\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/bb524831(v=vs.85).aspx)
 - <https://blogs.technet.microsoft.com/windowshpc/2015/02/02/how-to-compile-and-run-a-simple-ms-mpi-program/>
- Boost MPI
 - https://www.boost.org/doc/libs/1_68_0/doc/html/mpi.html
 - <https://theboostcpplibraries.com/boost.mpi>

MPI Hello Word!

Número de
processos

Computação

Universidade Federal de Pelotas

```
1 #include <stdio.h>
2 #include <string.h> /* For strlen */
3 #include <mpi.h> /* For MPI functions, etc */
4
5 const int MAX_STRING = 100;
6
7 int main(void) {
8     char greeting[MAX_STRING];
9     int comm_sz; /* Number of processes */
10    int my_rank; /* My process rank */
11
12    MPI_Init(NULL, NULL);
13    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
14    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
15
16    if (my_rank != 0) {
17        sprintf(greeting, "Greetings from process %d of %d!",
18                my_rank, comm_sz);
19        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
20                 MPI_COMM_WORLD);
21    } else {
22        printf("Greetings from process %d of %d!\n", my_rank, comm_sz);
23        for (int q = 1; q < comm_sz; q++) {
24            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
25                    0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26            printf("%s\n", greeting);
27        }
28    }
29
30    MPI_Finalize();
31    return 0;
32 } /* main */
```

`mpiexec -n 1 ./mpi_hello`

Greetings from process 0 of 1 !

`mpiexec -n 4 ./mpi_hello`

Greetings from process 0 of 4 !

Greetings from process 1 of 4 !

Greetings from process 2 of 4 !

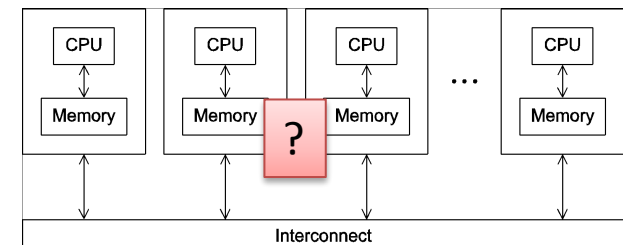
Greetings from process 3 of 4 !

Comunicadores

```
1 #include <stdio.h>
2 #include <string.h> /* For strlen */
3 #include <mpi.h> /* For MPI functions, etc */
4
5 const int MAX_STRING = 100;
6
7 int main(void) {
8     char greeting[MAX_STRING];
9     int comm_sz; /* Number of processes */
10    int my_rank; /* My process rank */
11
12    MPI_Init(NULL, NULL);
13    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
14    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
15
16    if (my_rank != 0) {
17        sprintf(greeting, "Greetings from process %d of %d!",
18                my_rank, comm_sz);
19        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
20                 MPI_COMM_WORLD);
21    } else {
22        printf("Greetings from process %d of %d!\n", my_rank, comm_sz);
23        for (int q = 1; q < comm_sz; q++) {
24            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
25                    0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26            printf("%s\n", greeting);
27        }
28    }
29
30    MPI_Finalize();
31    return 0;
32 } /* main */
```

- Uma coleção de processos que podem mandar mensagens entre si
- MPI_Init define o comunicador que consiste em todos os processos criados quando o programa começa

MPI_COMM_WORLD



Comunicação

```
int MPI_Send(
```

```
void*
```

```
int
```

```
MPI_Datatype
```

```
int
```

```
int
```

```
MPI_Comm
```

```
msg_buf_p
```

```
msg_size
```

```
msg_type
```

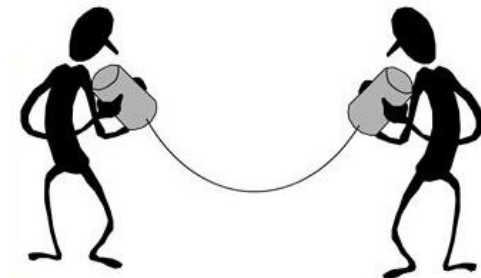
```
dest
```

```
tag
```

```
communicator
```

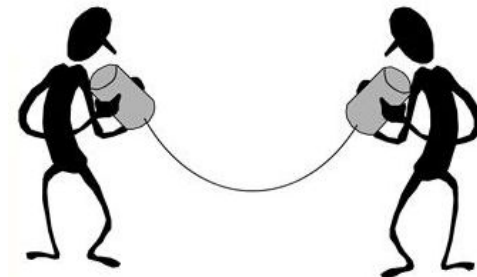


MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG	signed long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	



Comunicação

```
int MPI_Recv(  
    void*          msg_buf_p      /* out */,  
    int           buf_size        /* in  */,  
    MPI_Datatype   buf_type       /* in  */,  
    int           source          /* in  */,  
    int           tag             /* in  */,  
  
    MPI_Comm       communicator    /* in  */,  
    MPI_Status*    status_p        /* out */);
```



Remetente e destinatário

```
1 #include <stdio.h>
2 #include <string.h> /* For strlen */
3 #include <mpi.h> /* For MPI functions, etc */
4
5 const int MAX_STRING = 100;
6
7 int main(void) {
8     char greeting[MAX_STRING];
9     int comm_sz; /* Number of processes */
10    int my_rank; /* My process rank */
11
12    MPI_Init(NULL, NULL);
13    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
14    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
15
16    if (my_rank != 0) {
17        sprintf(greeting, "Greetings from process %d of %d!",
18                my_rank, comm_sz);
19        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
20                MPI_COMM_WORLD);
21    } else {
22        printf("Greetings from process %d of %d!\n", my_rank, comm_sz);
23        for (int q = 1; q < comm_sz; q++) {
24            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
25                    0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26            printf("%s\n", greeting);
27        }
28    }
29
30    MPI_Finalize();
31    return 0;
32 } /* main */
```

type, dest, send_tag,

r

PI_Recv

dest = r

type, src, recv_tag,

q

Ping-Pong



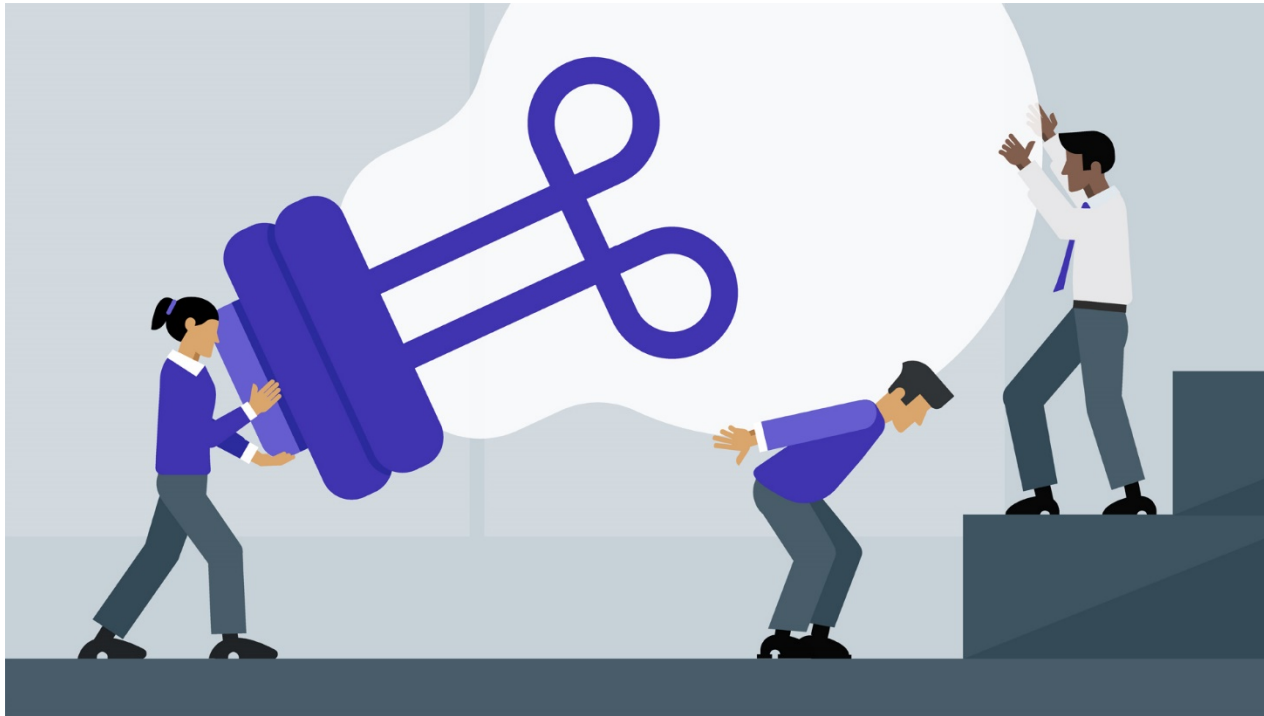
Printf de onde está a bola

Anel



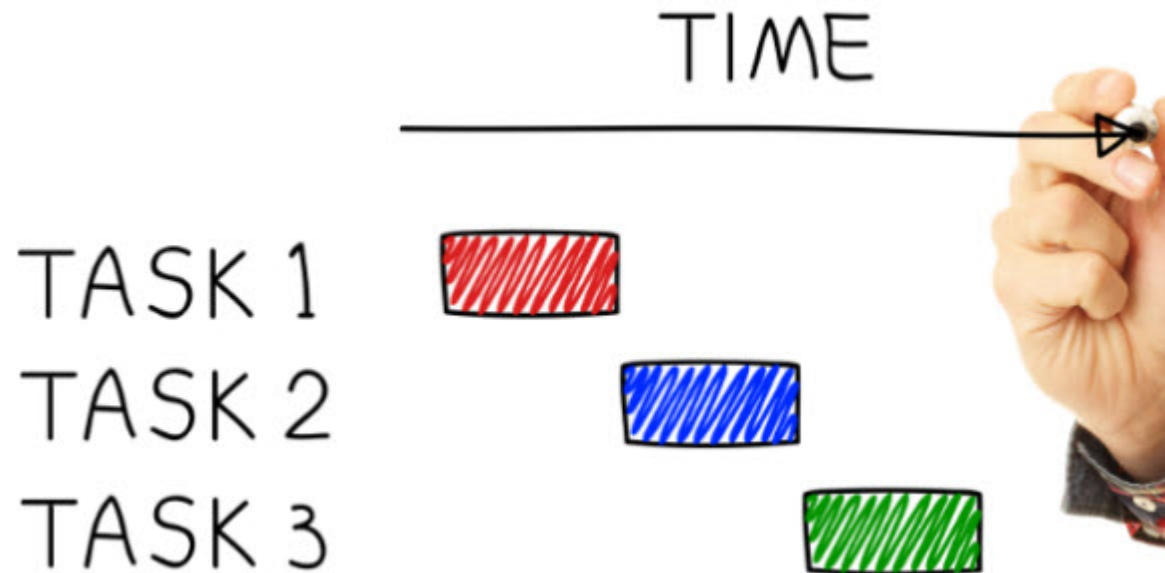
Printf de onde está a bola

Trabalho em equipe



Cada um precisa escolher algo diferente. Postar no tópico dessa tarefa no ava o que vai implementar. Na próxima segunda precisa explicar para todos o que fez.

Tempo



Apresentar quanto tempo cada etapa das tarefas anteriores consumiu.
Quanto tempo cada processo ficou aguardando ou trabalhando, etc...

Tarefas

1. <http://mpitutorial.com>
2. Fazer as 4 tarefas dos slides anteriores
3. Colocar no ava conforme for terminando
 1. No fórum tem um tópico pra cada