Guia Completo - `mpi4py` (Python MPI Interface)

O que é `mpi4py`?

É uma interface de alto desempenho que conecta Python ao padrão MPI (Message Passing Interface).Permite comunicação entre múltiplos processos (não threads!) seja em um cluster ou numa máquina multi-core. Usa bindings em C para performance máxima.

Conceito Central

O comunicador principal é: `MPI.COMM_WORLD`

`COMM_WORLD` inclui todos os processos que começaram a execução.

Principais Objetos ou Métodos/Funções desta lib

MPI.COMM_WORLD - É o comunicador padrão que engloba todos os processos.

Exemplo**:** comm = MPI.COMM_WORLD

comm.Get_size() - Retorna o número total de processos no comunicador.

Exemplo:

```
size = comm.Get_size()

print(f"Total de processos: {size}"
```

comm.Get_rank() - Retorna o rank (ID) do processo atual.

Exemplo:

```
rank = comm.Get_rank()

print(f"Meu rank: {rank}")
```

comm.send(data, dest, tag=0) - Envia dados para o processo `dest`.

Parâmetro	Descrição
data	Dados que você quer enviar (qualquer objeto serializável pelo Python).
dest	Rank do processo destino.
tag	(opcional) Etiqueta para diferenciar mensagens.

Exemplo:

```
if rank == 0:

    comm.send("Mensagem secreta", dest=1)
```

comm.recv(source = MPI.ANY_SOURCE, tag = MPI.ANY_TAG) - Recebe dados de outro processo.

Parâmetro	Descrição
source	Rank do processo origem (ou **`MPI.ANY_SOURCE`** para aceitar de qualquer um).
tag	(opcional) Etiqueta para filtrar mensagens.

Exemplo:

```
if rank == 1:

    data = comm.recv(source=0)

    print(f"Processo 1 recebeu: {data}")
```

comm.bcast(data, root) - Broadcast: o `root` envia o `data` para todos os outros processos.

Parâmetro	Descrição
data	Dados a enviar (no processo root).
root	Rank do processo que inicia o broadcast.

Exemplo:

```
data = None
    if rank == 0:
        data = {"chave": "valor"}
    data = comm.bcast(data, root=0)
    print(f"Processo {rank} recebeu: {data}")
```

comm.scatter(data, root) - Divide uma lista de dados entre os processos.

Parâmetro	Descrição
data	Lista ou array com os dados (somente no root).
root	Rank do processo que envia.

EXEMPLO:

```
if rank == 0:
    data = [i for i in range(size)]  # exemplo: [0,1,2,3]
else:
    data = None
    part = comm.scatter(data, root=0)
print(f"Processo {rank} recebeu {part}")
```

comm.gather(data, root) **- Coleta dados de todos os processos no `root`.**

Parâmetro	Descrição
data	Dado de cada processo.
root	Rank que vai receber todos os dados.

Exemplo:

```
part = rank * 2

collected = comm.gather(part, root=0)

if rank == 0:

    print(f"Root recebeu: {collected}")
```

comm.reduce(data , op = MPI.SUM, root = 0) - Reduz valores de todos os processos (ex: soma, máximo, mínimo) e envia para `root`.

Parâmetro	Descrição
data	Dado de cada processo.
op	Operação (default: soma).
root	Processo que recebe o resultado.

Exemplo:

```
local_sum = rank + 1  # cada processo calcula algo

total = comm.reduce(local_sum, op=MPI.SUM, root=0)

if rank == 0:

    print(f"Resultado final: {total}")
```

Comunicação Síncrona ou Assíncrona

send/recv são bloqueantes: o processo para e espera terminar.

Para comunicações não bloqueantes (executar em background):

> **`isend`** e **`irecv`** (returnam objetos de requisição que você pode controlar).

comm.Barrier() - Sincroniza todos os processos MPI. Nenhum processo pode ultrapassar a `Barrier` até que TODOS os outros também tenham chegado nela.

Exemplo:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

print(f"Processo {rank} pronto!")
comm.Barrier()
if rank == 0:
    print("Todos os processos sincronizados. Continuando execução...")
```

Serve para: **Garantir que todos estão no mesmo ponto da execução** antes de continuar (checkpoint). Evita problemas de **corrida** em execução paralela.

Importante: Se 1 processo travar antes da `Barrier`, todos os outros ficam esperando eternamente!

comm.Isend(data, dest, tag=0) - Inicia um envio de dados de maneira não bloqueante. Retorna um objeto `Request` para você acompanhar/completar depois.

Parâmetro	Tipo	Descrição
data	Qualquer objeto	Dados que serão

	serializável	enviados.
dest	int	Rank do processo de destino..
tag	int (opcional)	Identificador da mensagem (padrão 0). Permite distinguir mensagens diferentes.

EXEMPLO:

```
req = comm.Isend([dados, MPI.INT], dest=1, tag=55)
# Continua processamento
req.Wait()  # Garante que a mensagem foi enviada
```

comm.Irecv(buf, source = MPI.ANY_SOURCE, tag = MPI.ANY_TAG) - Inicia um recebimento de dados de maneira não bloqueante. Retorna também um objeto `Request`.

Parâmetro	Tipo	Descrição
buf	Qualquer objeto serializável	Onde armazenar os dados recebidos..
source	int (opcional)	Rank do processo remetente (padrão: qualquer fonte)..
tag	int (opcional)	Identificador da mensagem (padrão:

		qualquer tag).. Permite distinguir mensagens diferentes.

Exemplo:

```
dados = bytearray(100)  # Reservar espaço para receber
req = comm.Irecv([dados, MPI.BYTE], source=0, tag=55)
# Faz outras coisas
dados_recebidos = req.wait()  # Espera finalização
```

comm.Allgather(sendobj, recvobj=None) - Cada processo envia seus dados para todos os outros. Todos ficam com uma cópia completa dos dados de todos.

Parâmetro	Tipo	Descrição
sendobj	qualquer tipo	O dado local que cada processo quer enviar.
recvobj	lista (opcional)	Onde armazenar todos os dados coletados.

Exemplo:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
```

```
rank = comm.Get_rank()

dados = rank * 10  # Cada processo gera um número

# Todos recebem todos
resultado = comm.allgather(dados)

print(f"Processo {rank} recebeu {resultado}")
```

comm.Allreduce(sendobj, op=MPI.SUM) - Combina os dados de todos os processos usando uma operação de redução (soma, máximo, mínimo, etc). Todos os processos recebem o resultado final.

Parâmetro	Tipo	Descrição
sendobj	qualquer tipo	Dado local para reduzir.
op	operação MPI	Tipo de operação: `MPI.SUM`, `MPI.MAX`, `MPI.MIN`, etc.

Exemplo:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

valor = rank + 1  # Exemplo: processos 0,1,2 => valores 1,2,3

# Soma todos os valores
resultado = comm.allreduce(valor, op=MPI.SUM)

print(f"Processo {rank} sabe que a soma total é {resultado}")
```

Exemplo Completo (Misto de Comunicação)

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Root envia mensagem para todos
msg = comm.bcast(f"Oi, eu sou o processo {rank}", root=0)

print(f"Processo {rank} recebeu: {msg}")

# Cada processo calcula um valor
value = rank * 2

# Todos os valores somados
total = comm.reduce(value, op=MPI.SUM, root=0)

if rank == 0:
    print(f"Soma de todos os valores: {total}")
```

Exemplo Avançado: Multiples Envios e Recebimentos Assíncronos

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

if size < 2:
    raise Exception("Precisa de pelo menos 2 processos!")

# Cada processo envia algo para o próximo processo (em ciclo)
next_rank = (rank + 1) % size
prev_rank = (rank - 1) % size

# Enviar e receber ao mesmo tempo
send_req = comm.isend(f"Olá do processo {rank}", dest=next_rank)
recv_req = comm.irecv(source=prev_rank)

# Computação pesada enquanto comunica
for i in range(3):
    print(f"Processo {rank} fazendo trabalho {i}")
    time.sleep(0.5)

# Finalizar comunicação
send_req.Wait()
msg = recv_req.wait()

print(f"Processo {rank} recebeu mensagem: {msg}")
```