- O MPI como um padrão e um interface completa para passagem de mensagens em programação paralela;
- Modelo do MPI:
  Se baseia em 3 entidades:
    - Communicator - Grupo de processos com habilidade de comunicar-se uns com os outros;
    - Rank - identificação de cada processo;
    - Tag - Identificação de uma mensagem;

MPI - Principais funções

- Comm.Get_rank()
- Retorna o rank de um processo em um communicator;
- A cada processo é atribuído um número incremental começando em zero;
- Usado para identificar um processo durante o envio e recepção de mensagens.

MPI Exemplo

```
#hello.py
from mpi4py import MPI
comm = MPI.COMM_world
rank = comm.Get_rank()
print("Olá do processo", rank)
```

- Envio e recepção de mensagens MPI
- Dados são empacotados e colocados em um buffer (envelope);
- Um processo deve confirmar que deseja receber uma mensagem;
- Mensagens são identificadas através de tags;
- As mensagens recebidas com uma tag diferente da esperada são guardadas no buffer da rede.
- `data=comm.recv(int source, int tag)`
    - data - onde os dados recebidos são armazenados;
    - source - rank do processo de origem;
    - tag - identificação da mensagem;

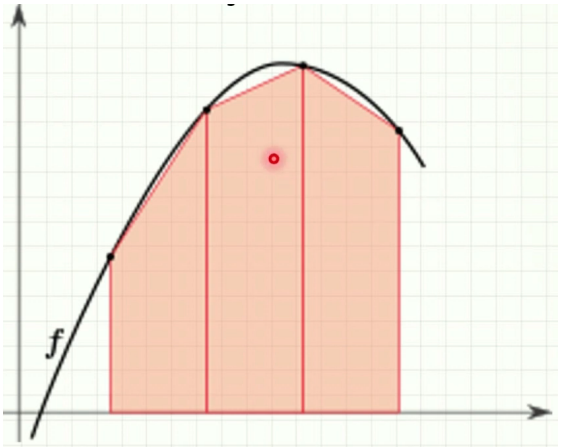
Exemplo

```
import numpy
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()

randNum = numpy.zeros(1)
if rank == 1:
        randNum = numpy.random.random_sample(1)
        print("Processo", rank, "gerou o número", randNum[0])
        comm.Send(randNum, dest=0)
if rank == 0:
        print("Processo", rank, "antes de receber o número", randNum[0])
        randNum=comm.Recv(source=1)
        print("Processo", rank, "recebeu o número", randNum[0])
```

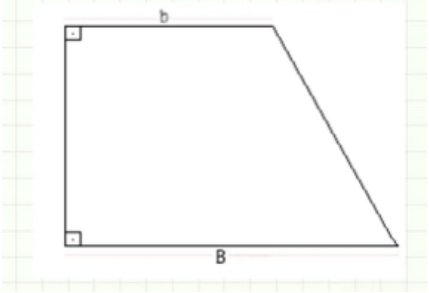
Exemplo: Método dos trapézios

Utilizado para o cálculo aproximado da integral de uma função.

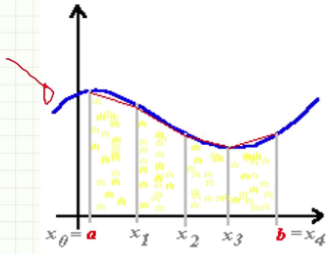


Área do trapézio:

$$area = \frac{(B+b)h}{2}$$



- $A_{total}=\frac{h}{2}\left[f(a)+2x\left(f(x_1)+f(x_2)+\cdots+f(x_k)\right)+f(b)\right]$
- $A_{total}=h\left[\frac{f(a)+f(b)}{2}+\left(f(x_1)+f(x_2)+\cdots+f(x_k)\right)\right]$
- $h=\frac{b-a}{n}$



Código exemplo:

```
def integrateRange(a, b, n):
    integral = (f(a) + f(b))/2.0
    vet=numpy.linspace(a,b,n+1)
    for x in range(1,len(vet)-1):
        integral = integral + f(vet[x])
    integral = integral* (b-a)/n
    return integral
```

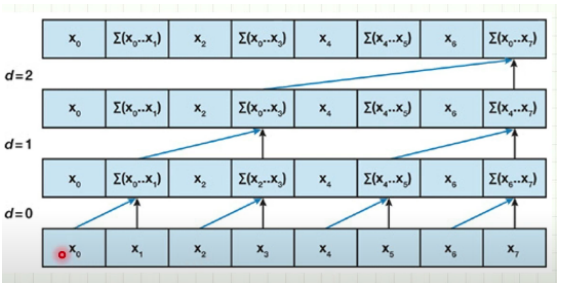
Distribuindo o cálculo entre os nós

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
h = (b-a)/n
local_n = n/size
local_a = a + rank*local_n*h
local_b = local_a + local_n*h
integral = numpy.zeros(1)
recv_buffer = numpy.zeros(1)
integral[0] = integrateRange(local_a, local_b, local_n)
if rank == 0:
    total = integral[0]
    for i in range(1, size):
        comm.Recv(recv_buffer, ANY_SOURCE)
        total += recv_buffer[0]
else:
    comm.Send(integral)
if comm.rank == 0:
    print("Com n =", n, "trapézios, a integral estimada de", a, "a", b, "é", total)
```

Comunicação coletiva

- Problema: Os resultados de todos nós são somados por um único processo.
- Comunicação entre processos é custosa;
- Métodos Reduce e Broadcast para otimizar a comunicação;

Método Reduce



Método dos trapézios com Reduce

```
local_a = a + rank*local_n*h
local_b = local_a + local_n*h
integral = numpy.zeros(1)
total = numpy.zeros(1)
integral[0] = integrateRange(local_a, local_b, local_n)
comm.Reduce(integral, total, op=MPI.SUM, root=0)
if comm.rank == 0:
    print "Com n =", n, "trapézios,  a integral estimada de"\ , a, "até", b, "é", total
```

Comm.Reduce()

Comm.**Reduce**(*sendbuf, recvbuf, Op op = MPI.SUM, root = 0*)

Reduces values on all processes to a single value onto the *root* process.

Parameters:
- **Comm** (*MPI comm*) – communicator we wish to query
- **sendbuf** (*choice*) – address of send buffer
- **recvbuf** (*choice*) – address of receive buffer (only significan
- **op** (*handle*) – reduce operation
- **root** (*int*) – rank of root operation

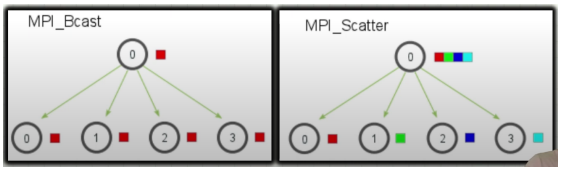
Operações suportadas

Name	Meaning
MPI.MAX	maximum
MPI.MIN	minimum
MPI.SUM	sum
MPI.PROD	product
MPI.LAND	logical and
MPI.BAND	bit-wise and
MPI.LOR	logical or
MPI.BOR	bit-wise or
MPI.LXOR	logical xor
MPI.BXOR	bit-wise xor
MPI.MAXLOC	max value and location
MPI.MINLOC	min value and location

Bcast X Scatter

Métodos de broadcast

- Bcast - Envia o mesmo dado a todos os processos envolvidos;
- Scatter - Envia pedaços de um vetor para diferentes processos;



Bcast e Scatter

Comm.**Bcast**(*buf, root=0*)

Broadcasts a message from the process with rank "root" to all other processes of the group.

Parameters:
- **Comm** (*MPI comm*) – communicator across which to broadcast
- **buf** (*choice*) – buffer
- **root** (*int*) – rank of root operation

Comm.**Scatter**(*sendbuf, recvbuf, root*)

Sends data from one process to all other processes in a communicator (Length of r
Scatterv)

Parameters:
- **sendbuf** (*choice*) – address of send buffer (significant only at root)
- **recvbuf** (*choice*) – address of receive buffer
- **root** (*int*) – rank of sending process

Exemplo Bcast()

```
import numpy
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()

#intialize
rand_num = numpy.zeros(1)

if rank == 0:
        rand_num[0] = numpy.random.uniform(0)

comm.Bcast(rand_num, root = 0)
print "Process", rank, "has the number", rand_num
```

Exemplo Scatter()

```
import numpy
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
LENGTH = 3

if rank == 0:
    x = numpy.linspace(1,size*LENGTH,size*LENGTH)
else:
    x = None
x_local = numpy.zeros(LENGTH)

comm.Scatter(x, x_local, root=0)

print("process", rank, "x:", x)
print("process", rank, "x_local:", x_local)
```

O produto ponto

- $\sum_{k}^{n} = 1 u_k v_k$
- Multiplicação de dois vetores;
- Versão paralela
    - Divide os vetores entre os processos;

- Executa o produto em cada processo;
- Usa o Reduce com MPI_SUM para coletar os resultados.