2019/20

Programação Paralela e Distribuída

Prática 6: MPI (for Python) – Operações básicas

(https://mpitutorial.com/tutorials/)

1 – Instalar MPI
Open MPI (https://www.open-mpi.org/) ou
Microsoft MPI (https://docs.microsoft.com/en-us/message-passing-interface/microsoft-mpi)

2 – Instalar mpi4py: https://mpi4py.readthedocs.io/en/stable/

3 – Implementar um programa em que todos os processos diferentes do processo 0, enviam uma mensagem para o processo 0 com a mensagem "Olá sou o processo nº X".
a) O que acontece se executa com apenas um processo?

4 - Escreva um programa onde um processo com rank K envie uma mensagem ao processo com rank (K+1)%p. A mensagem é a string "Olá, do processo K", e não deve utilizar wildcards. Execute o programa com vários processos.

5 – No programa que se segue, para comunicar entre processos, usar apenas as instruções send e recv. O processo 0 deve criar uma mensagem com uma string que contenha o seu nome. Esta mensagem deve ser passada entre os processadores arranjados logicamente num anel $P_0,P_1,..P_{n-1}$ até a mensagem chegar novamente ao processo com rank zero onde a mensagem será impressa no ecrã.

6 – Ping-Pong.
Escrever um programa para fazer uma estimativa da latência, do tempo de transmissão e largura de banda (bandwidth). ***T*comm = *T*startup + *n T*data** onde,

*T*comm é o tempo total de comunicação,
*T*startup é o tempo de *startup tim*e (*latency)*
T data é tempo de transmissão por byte
Bandwidth - velocidade de transmissão medida em Mega bytes per segundo é o inverso de Tdata

Usar o método de "ping-pong": - um processo envia uma mensagem de vários tamanhos a outro processo que depois de a receber envia-a de volta para o primeiro processo.
Para medir o tempo de transmissão utilizar a função MPI.Wtime(). O programa deverá ter a estrutura seguinte:
a) – Medir o custo adicional (overhead) das chamadas do MPI.Wtime().
inicio = MPI.Wtime() /* sem nada */ fim = MPI.Wtime() tempo = fim - inicio

b) – Ping Pong
Declarar um array de floats de tamanho suficiente. Não é preciso inicializar.
Para cada tamanho (n) de mensagem (número de bytes a enviar):

No rank 0:

Chamar a função MPI Barrier() para que os dois processos sincronizem;

Iniciar o relógio;
Enviar mensagem de n floats para processo 1;
Receber mensagem de n floats do processo 1;
Fim do relógio e calcular o tempo demorado;

No rank 1:

Chamar a função MPI Barrier() para que os dois processos sincronizem;
Receber mensagem de n floats do processo 0;
Enviar mensagem de n floats para processo 0;

Notas:

- Repetir o processo várias vezes e calcular o tempo médio.
- Precisa de saber o tamanho em bytes de um float !
- O programa deve executar em modo local (e distribuído)-

7 - Construa um programa para calcular a média de um array de valores, usando as operações MPI Scatter e MPI Gather em mpi4py. Abaixo tem o algoritmo esquematizado na linguagem c.

```
if (world rank == 0) {
  rand_nums = create_rand_nums(elements_per_proc * world_size);
}
// Create a buffer that will hold a subset of the random numbers
float *sub rand nums = malloc(sizeof(float) * elements per proc);
// Scatter the random numbers to all processes
MPI_Scatter(rand_nums, elements_per_proc, MPI_FLOAT, sub_rand_nums,
            elements_per_proc, MPI_FLOAT, 0, MPI_COMM_WORLD);
// Compute the average of your subset
float sub avg = compute avg(sub rand nums, elements per proc);
// Gather all partial averages down to the root process
float *sub avgs = NULL;
if (world_rank == 0) {
  sub_avgs = malloc(sizeof(float) * world_size);
}
MPI Gather(&sub avg, 1, MPI FLOAT, sub avgs, 1, MPI FLOAT, 0,
           MPI COMM WORLD);
// Compute the total average of all numbers.
if (world_rank == 0) {
  float avg = compute_avg(sub_avgs, world_size);
}
```

8 – Construa um programa para calcular a média de um array de valores, usando MPI Reduce em mpi4py. Abaixo tem o algoritmo esquematizado na linguagem c.

```
float *rand_nums = NULL;
rand_nums = create_rand_nums(num_elements_per_proc);

// Sum the numbers locally
float local_sum = 0;
int i;
for (i = 0; i < num_elements_per_proc; i++) {
  local_sum += rand_nums[i];
}

// Print the random numbers on each process
printf("Local sum for process %d - %f, avg = %f\n",
       world rank, local sum, local sum / num elements per proc);

// Reduce all of the local sums into the global sum
float global sum;
MPI_Reduce(&local_sum, &global_sum, 1, MPI_FLOAT, MPI_SUM, 0,
           MPI_COMM_WORLD);

// Print the result
if (world rank == 0) {
  printf("Total sum = %f, avg = %f\n", global_sum,
         global_sum / (world_size * num_elements_per_proc));
}
```