Message Passing Interface (MPI)

Patrick Silva Ferraz[1]

[1]Ciência da Computação – Universidade Estadual de Santa Cruz (UESC)
45662-900 – Ilhéus – BA – Brasil

patrick.ferraz@outlook.com

Resumo. *Este relatório introduz o conceito de MPI e descreve o processo de instalação, configuração e utilização da biblioteca OpenMPI em duas máquinas virtuais Ubuntu 16.04.4 LTS de 32bits.*

1. Introdução

O Message Passing Interface (MPI) é um padrão para comunicação de dados em computação paralela. O principal objetivo é prover um amplo padrão para escrever programas com passagem de mensagens de forma prática, portátil, eficiente e flexível. Duas palavras chaves devem ser apresentadas:

Processo: Quando o programa é executado ele é "quebrado em partes", cada parte é chamado de processo. Estes processos podem ser executados em uma únicas máquina ou múltiplas.

Rank: Cada processo, após iniciado, possui uma identificação única atribuída pelo sistema. Essa identificação é contínua representada por um número inteiro, começando de zero até N-1, onde N é o número de processos. Cada processo possui um rank, sendo utilizando para enviar e receber mensagens.

2. Procedimento

Para a realização das etapas, foi criado duas máquinas virtuais Ubuntu 16.04.4 LTS (hosts) utilizando o VMware Workstation 12. Todos passos descritos abaixo foram executadas em ambos hosts.

A Figura 1 possui a exibição das máquinas virtuais rodando, com seus respectivos IPs.

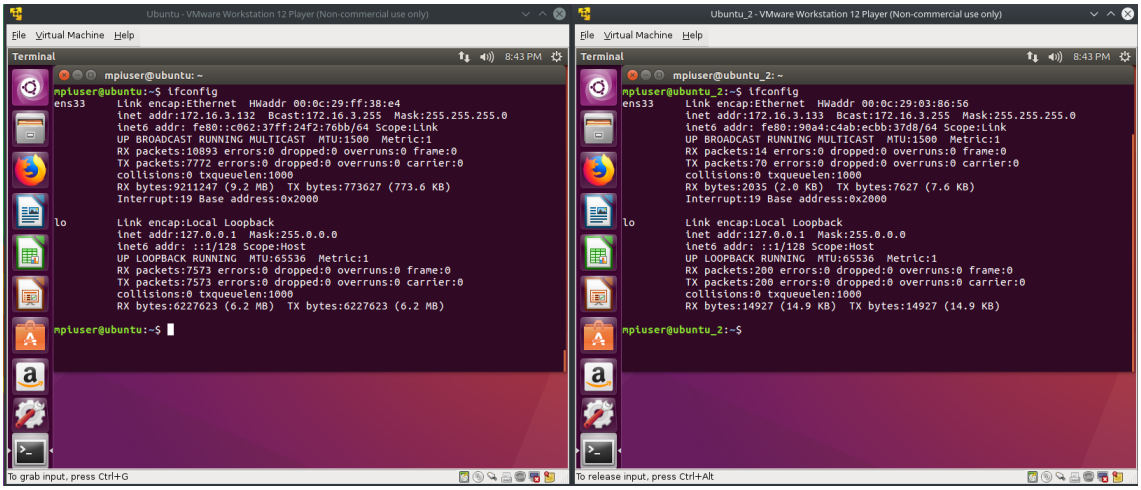

Figura 1. Máquinas virtuais Ubuntu 16.04.4 LTS (esquerda Ubuntu, direita Ubuntu_2).

1. Afim de facilitar a execução dos processos, foram atribuídos nomes para ambos hosts editando o arquivo **/etc/hosts** adicionando as seguintes linhas:

   172.16.3.132 ubuntu # máquina virtual 1

   172.16.3.133 ubuntu_2 # máquina virtual 2

   Desta forma as máquinas podem ser identificadas por nome ao invés de IP.

2. Em ambos hosts foram criados um novo usuário com nome **mpiuser** contendo a mesma senha:

   $ adduser mpiuser

3. Pois bem, é necessário a instalação do openmpi. Em alguns casos é necessário também instalar o compilador C/C++ caso suas distribuição não possua. O comando informado abaixo é utilizado para instalar o openmpi e algumas das suas dependências, assim como o ssh (responsável pela comunicação remota com os demais hosts).

   $ sudo apt-get install openmpi-bin openmpi-common openssh-client openssh-server libopenmpi1.10 libopenmpi-dev gcc g++

4. As próximas etapas devem ser realizadas no novo usuário criado, **mpiuser**.

5. Para a distribuição do processamento, é necessário que a comunicação ocorra entre os hosts de forma automática e confiável. Por conta disso foi gerado um par de chaves com a frase secreta “em branco” e este copiado para ambos hosts.

   $ ssh-keygen -t rsa # criação da chave

   $ ssh-copy-id -i ~/.ssh/id_rsa.pub mpiuser@[ubuntu/ubuntu_2]

   $ /etc/init.d/ssh restart

6. O teste de conexão pode ser realizado utilizando o comando abaixo dependendo

do host que estiver em uso:

$ ssh mpiuser@[ubuntu/ubuntu_2]

7. Foi criado o arquivo my_target contendo o endereço (nome) das máquinas hosts que realizarão a distribuição dos processos.

   $ echo ubuntu > my_target; echo ubuntu_2 >> my_target

8. Para realizar os testes, foi criado e compilado o arquivo **cpi.c**, que calcula pi, em ambos hosts, na pasta home do usuário. Abaixo está descrito o código executado e logo após o comando para compilação e execução.

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );
double f( double a )
{
   return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
   int done = 0, n, myid, numprocs, i;
   double PI25DT = 3.14159265358979323846264 3;
   double mypi, pi, h, sum, x;
   double startwtime = 0.0, endwtime;
   int  namelen;
   char processor_name[MPI_MAX_PROCESSOR_NAME];

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   MPI_Get_processor_name(processor_name,&namelen);

   fprintf(stderr,"Process %d on %s\n",
               myid, processor_name);

   n = 0;
   while (!done)
   {
      if (myid == 0)
      {
/*
         printf("Enter the number of intervals: (0 quits) ");
         scanf("%d",&n);
*/
               if (n==0) n=100; else n=0;

               startwtime = MPI_Wtime();
```

```
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
           done = 1;
        else
        {
           h   = 1.0 / (double) n;
           sum = 0.0;
           for (i = myid + 1; i <= n; i += numprocs)
           {
              x = h * ((double)i - 0.5);
              sum += f(x);
           }
           mypi = h * sum;

           MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

           if (myid == 0)
             {
              printf("pi is approximately %.16f, Error is %.16f\n",
                    pi, fabs(pi - PI25DT));
                         endwtime = MPI_Wtime();
                         printf("wall clock time = %f\n",
                               endwtime-startwtime);
              }
        }
     }
     MPI_Finalize();

     return 0;
}
```

```
$ mpicc -o runpi cpi.c                          # compilação
$ mpirun --hostfile my_target -np 10 ./runpi    # execução
```

Atenção: Para que a execução ocorra de forma distribuída, é necessário que exista o arquivo compilado em ambos hosts.

3. Resultados

As figuras seguintes exibem os resultados obtidos ao realizar a distribuição dos 10 processos para os hosts (ubuntu e ubuntu_2).

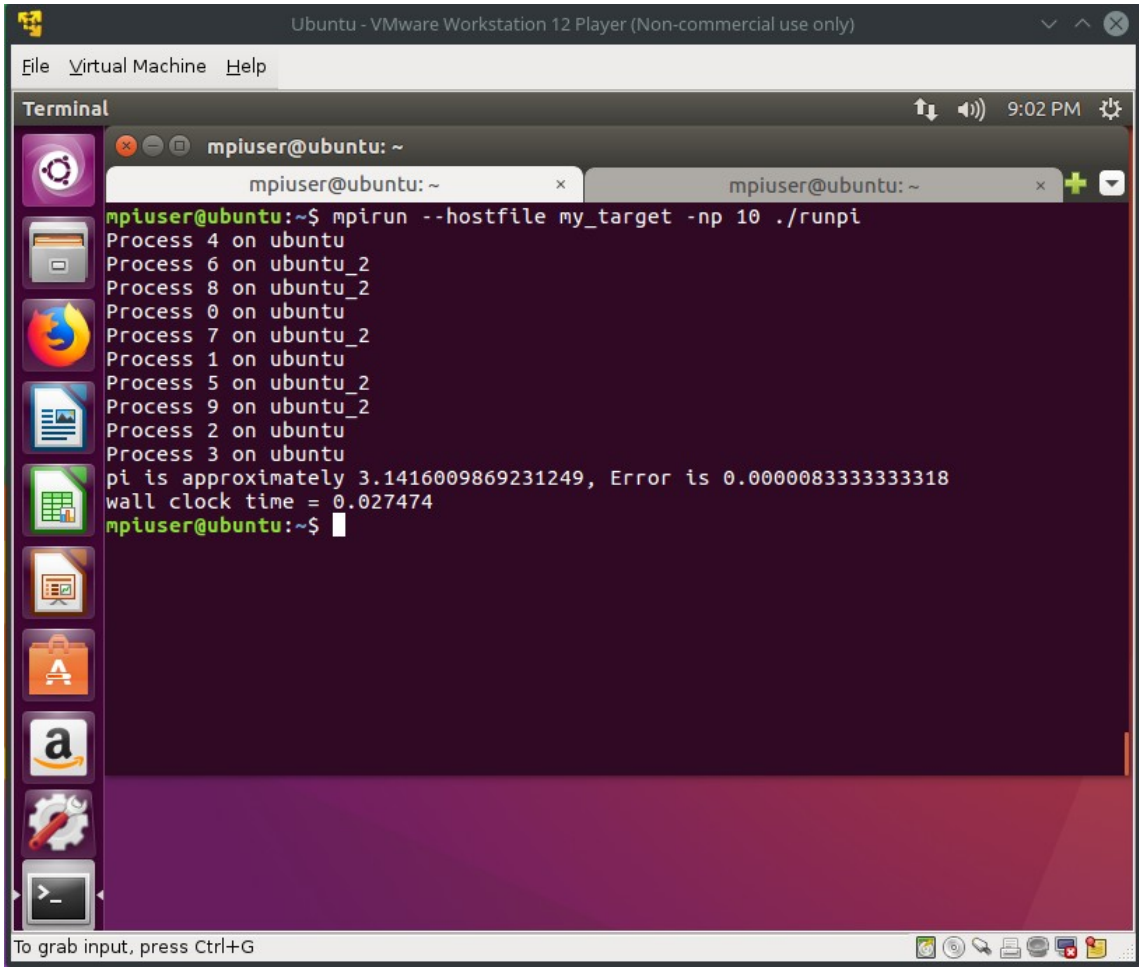

Figura 2. Execução de runcpi no host Ubuntu e Ubuntu_2.

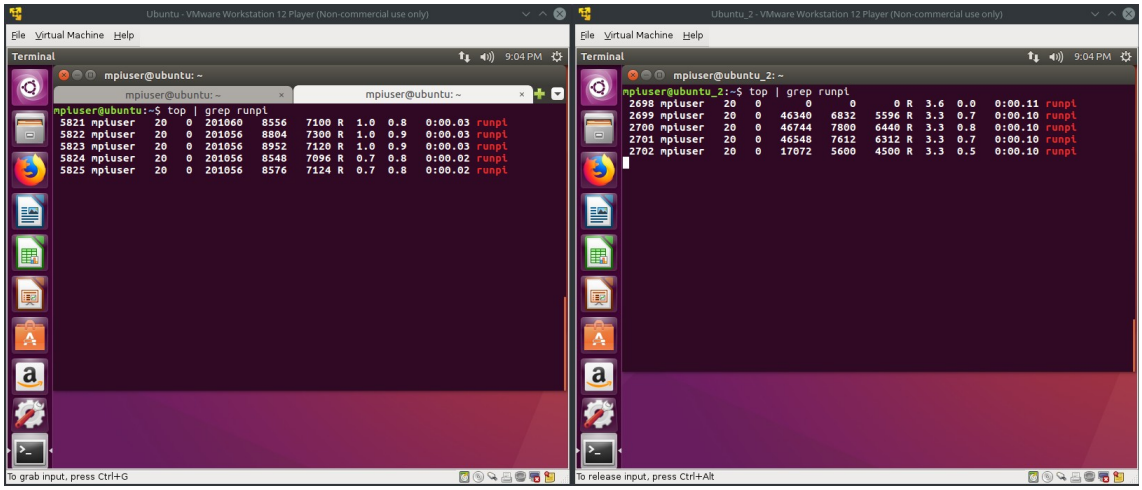

Figura 3. Exibição dos processos sendo executados em Ubuntu (5 processos) e Ubuntu_2 (5 processos).