Introdução à programação paralela

Gabriel Martins de Miranda – 13/0111350

1. Título do capítulo

Greetings! - Programação Paralela com MPI.

2. Objetivo do capítulo

O capítulo objetiva apresentar um programa simples de comunicação entre processos utilizando a biblioteca mpi.h, mostrando funções básicas e focando nas de envio, MPI_Send, e recebimento, MPI_Recv.

3. Resumo do capítulo

É apresentada a biblioteca de programação paralela em C, MPI. O tipo de programação usada , SPMD, gera executáveis para cada processo de um único fonte, em que neste especificamos qual processo irá executar através de um sistema de rank. Todo identificador começa com **MPI_**

Para incluir a biblioteca:

```
#include <mpi.h>
```

Para inicializar os recursos:

```
int MPI_Init(int *argc, char **argv[]);
```

Para finalizar os recursos:

```
int MPI_Finalize(void);
```

Para contabilizar processos envolvidos:

```
int MPI_Comm_size(MPI_Comm comm, int *nproc);
```

Para encontrar rank atual:

```
int MPI_Comm_rank(MPI_Comm comm, int *myrank);
```

Para enviar mensagem:

```
int MPI_Send(
    void *message,              /* ref para dado a ser enviado */
    int count,                  /* n de itens na mensagem */
    MPI_Datatype datatype,      /* tipo de item na mensagem */
    int dest,                   /* rank do processo a receber a mensagem */
    int tag,                    /* identificador especifico do programa */
    MPI_Comm comm               /* comunicador */
);
```

Para receber mensagem:

```
int MPI_Recv(
        void *message,          /* referencia para buffer de dados recebido */
        int count,              /* n de itens a ser recebido */
        MPI_Datatype datatype,  /* tipo do item a ser recebido */
        int source,             /* rank do processo remetente */
        int tag,                /* identificador especifico do programa */
        MPI_Comm comm           /* comunicador */
        MPI_Status *status      /* armazena info da mensagem recebida */
);
```

Sendo a mensagem constituída de dados e envelope.

4. Solução dos exercícios

Foi usado o MPICH, implementação de alta performance e portabilidade do MPI.

1. Create a C source file containing the "Greetings!" program. Find out how to compile it and run it on different numbers of processors. What is the output if the program is run with only one process? How many processors can you use?

Para compilar: ***$ mpicc greetings.c -o greetings***

Para executar: ***$ mpiexec -n 4 greetings****, onde -n representa o número de processos criados, sendo cada um executado por um processador (teoricamente). No exemplo são usados 4 processos.*

Executando com n=1, ***$ mpiexec -n 1 greetings****, não há saída, pois utilizando-se apenas um processador não é possível haver comunicação entre processos de processadores diferentes.*

Posso utilizar o número de processadores que quiser, desde que esteja disponível. No caso do meu computador pessoal, 4 processadores são utilizados.

2. Modify the "Greetings!" program so that it uses wildcards in the receives for both source and tag. Is there any difference in the output of the program?

Para o uso de wildcards, a função MPI_Recv foi alterada, sendo as modificações dadas por ***source => MPI_ANY_SOURCE*** *e* ***tag => MPI_ANY_TAG****.*

A saída do programa foi a mesma, porém percebeu-se que a ordem da recepção das mensagens alterou-se para uma forma aleatória, sendo que no caso do exercício anterior as mensagens eram recebidas em ordem crescente de processos.

3. Try modifying some of the parameters to MPI_Send and MPI_Recv (e.g., count, datatype, source, dest). What happens when you run your program? Does it crash? Does it hang (i.e., stop in the middle of execution without crashing)?

Caso 1: MPI_Send com count => 8 e MPI_Recv com count => 4
saída: Programa abortado com erro.

Caso 2: MPI_Send com datatype => MPI_INT e MPI_Recv com datatype => MPI_CHAR
saída: Programa abortado com erro.

Caso 3: MPI_Send com dest => 1 e MPI_Recv com source => 0
saída: Nunca tem fim.

4. Modify the "Greetings!" program só that all the processes send a message to process p-1. On many parallel systems, every process can print to the screen of the terminal from which the program was started. Have process p-1 print the messages it receives. What happens? Can process p-1 print to the screen?

O processo p-1 consegue printar normalmente na tela.

5. Trabalhos de programação

1. Write a program in which process i sends a greeting to process (i+1)%p. (Be careful of how i calculates from whom it should receive!). Should process i send its message to process i+1 first and then receive the message from process i-1? Should it first receive and then send? Does it matter? What happens when the program is run on one processor?

Agora todos os processos conversam cosigo mesmos. Quando MPI_Send vem antes de MPI_Recv, os processadores conversam entre si em ordem aleatória, sendo que caso n=3, temos três linhas de saída. A execução não termina se MPI_Recv vem antes.

6. Conclusão

Através do capítulo foi possível entender a base de um programa paralelo, como cada instância é executada como um processo e cada processo executado pelos diferentes processadores do computador.

7. Referências consultadas

http://www.mpich.org/downloads/
https://www.mpich.org/documentation/guides/
https://www.sharcnet.ca/help/index.php/Getting_Started_with_MPI
MPICH User's Guide Version 3.2
http://www.mcs.anl.gov/research/projects/mpi/tutorial/

Obs.: As primeiras linhas nos fontes anexados indicam como compilar.