Programação com MPI

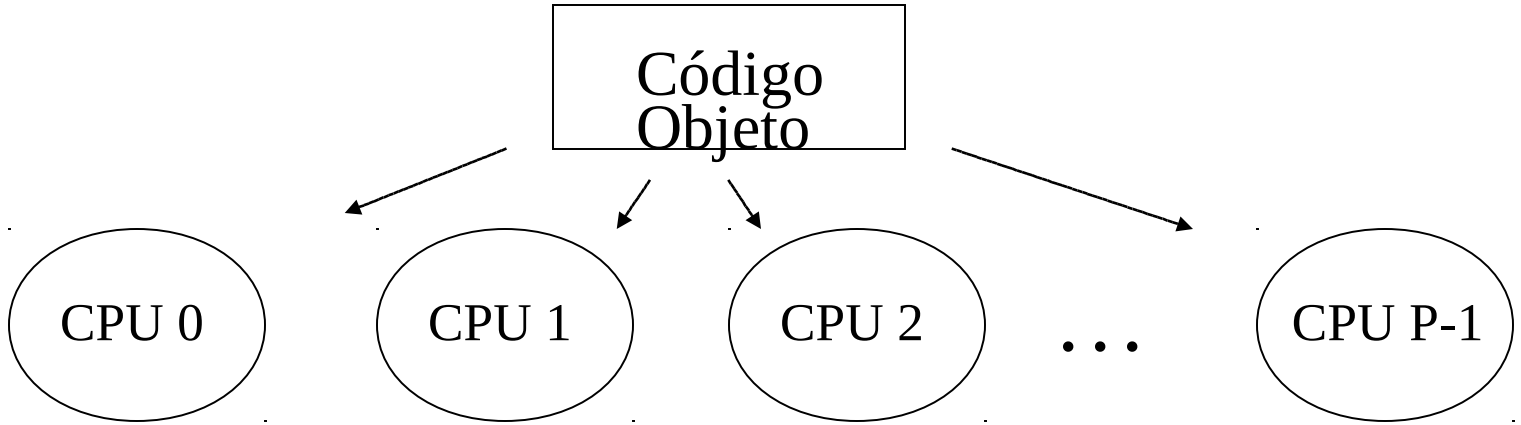
Tópicos:

- Modelo de Programação
- Funções de Ambiente
- Funções Básicas com Mensagens
- Exemplo de Programa com MPI
- Sumário

Referências:

- Pacheco,P.S. *Parallel Programming with MPI*, Morgan Kaufmann, San Francisco, 1997
- http://www.mcs.anl.gov/mpi

Modelo de Programação



- Número de CPU's (*size*): P (definido na hora da execução)
- Identificação de cada CPU (*rank*): 0, 1, ..., P-1
- Mesmo código objeto executado por todas as CPU's
- Todas as variáveis são locais

Modelo de Programação (cont.)

Observações:

- Cada CPU pode executar ou não alguns trechos: blocos com `if ( rank == ... ) { ... }`
- Modelo de Execução:
  - SPMD (***S**ingle-**P**rogram / **M**ultiple-**D**ata*)
- MPI: biblioteca de funções e definições
  - NÃO é uma nova linguagem !
- Funções de MPI: `MPI_Func(...)`
- Constantes em MPI: `MPI_CONST...`

Modelo de Programação (cont.)

- Estrutura geral de um programa em linguagem C :

```
#include ‘’mpi.h’’
. . .
main(int argc, char *argv[] ) {
 . . .
 MPI_Init(&argc,&argv);
 . . .   → trecho com possíveis chamadas a funções de MPI
 MPI_Finalize();
 . . .
}
```

Funções de Ambiente em MPI

- Verificação do número de CPU's:

```
MPI_Comm_size( MPI_Comm communicator, int* size )
```

- Verificação da identificação de cada CPU:

```
MPI_Comm_rank( MPI_Comm communicator, int* rank )
```

- Conceito de *Communicator*:
  - Grupo de CPU's que podem trocar mensagens entre si
  - Communicator especial: MPI_COMM_WORLD (todas as CPU's que estão executando o programa)

Funções Básicas com Mensagens

```
int MPI_Send( void*           buffer,
              int             count,
              MPI_Datatype    datatype,
              int             destination,
              int             tag,
              MPI_Comm        communicator )

int MPI_Recv( void*           buffer,
              int             count,
              MPI_Datatype    datatype,
              int             source,
              int             tag,
              MPI_Comm        communicator,
              MPI_Status*     status )
```

Funções Básicas com Mensagens (cont.)

- Parâmetros:
  - `buffer`: Endereço em memória da mensagem
  - `count`: Número de ítens na mensagem
  - `datatype`: tipo de cada ítem
  - `destination`: identificação da CPU de destino
  - `source`: identificação da CPU de destino de envio
    - possível “coringa”: MPI_ANY_SOURCE
  - `tag`: identificação do tipo
    - possível “coringa” para recv: MPI_ANY_TAG
  - `communicator`: grupo de CPU’s
  - `status`: Estrutura com valores sobre a msg recebida:
    - status -> MPI_SOURCE
    - status -> MPI_TAG
    - status -> MPI_ERROR

Funções Básicas com Mensagens (cont.)

- Semântica de `MPI_Send()` e `MPI_Recv()`:
  - Ambas são assíncronas → podem ser executadas pelas respectivas CPU's em instantes distintos
  - Ambas funcionam com bloqueio → retorno ao prog. principal somente quando `buffer` pode ser utilizado
    - MPI_Send(buffer,...) : ao retornar, já se pode reutilizar *buffer*
    - MPI_Recv(buffer...) : ao retornar, *buffer* já tem msg recebida
- Há outras funções para troca de mensagens, síncronas ou sem bloqueio

Exemplo de Programa com MPI

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"
main(int argc, char* argv[]) {
    int     my_rank;       /* rank of process      */
    int     p;             /* number of processes  */
    int     source;        /* rank of sender       */
    int     dest;          /* rank of receiver     */
    int     tag = 0;       /* tag for messages     */
    char    message[100];  /* storage for message  */
    MPI_Status  status;    /* return status for recv */
/* Start up MPI */
    MPI_Init(&argc, &argv);
/* Find out process rank  */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
/* Find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &p);
```

Exemplo de Programa (cont.)

```
    if (my_rank != 0) {
        /* Create message */
        sprintf(message, "Greetings from process %d!",
            my_rank);
        dest = 0;
        /* Use strlen+1 so that '\0' gets transmitted */
        MPI_Send(message, strlen(message)+1, MPI_CHAR,
            dest, tag, MPI_COMM_WORLD);
    } else { /* my_rank == 0 */
        for (source = 1; source < p; source++) {
            MPI_Recv(message, 100, MPI_CHAR, source, tag,
                MPI_COMM_WORLD, &status);
            printf("%s\n", message);
        }
    }
    /* Shut down MPI */
    MPI_Finalize();
} /* main */
```

Exemplo de Programa (cont.)

Compilação (usando MPICH):

```
> mpicc  -o cgreetings greetings.c
> mpif90 -o fgreetings greetings.f90
```

- Execução com 4 CPU's:

```
> mpiexec -n 4 greetings

 Greetings from process 1!
 Greetings from process 2!
 Greetings from process 3!
```

Exemplo de Programa (cont.)

Comando mpirun (opção -machinefile):

```
> mpirun -n 8 ./cgreetings
```

Comando mpiexec com MPD daemon:

```
> mpdboot
> mpiexec -np 8 ./cgreetings
> mpdallexit
```

Programas com MPI: Sumário

- Estrutura da maioria dos programas com MPI:

```
#include ‘’mpi.h’’
. . .
main(int argc, char *argv[] ) {
 . . .
 MPI_Init(&argc,&argv);
 . . .
 MPI_Comm_size (MPI_COMM_WORLD, &size)
 MPI_Comm_rank (MPI_COMM_WORLD, &rank)
 . . .
  { MPI_Send(...) , MPI_Recv(...) }
 . . .
 MPI_Finalize();
 . . .
}
```