MPI – *Message Passing Interface*

Avelino F. Zorzo

MPI

- *Message Passing Interface*
- Surgiu em 1992 para padronizar as bibliotecas de troca de mensagens
  - Estável; Eficiente; Portável
- Biblioteca de funções para C, FORTRAN e C++ executando em máquinas MIMD com organização de memória NORMA
- 129 funções com vários parâmetros e variantes
- Conjunto fixo de processos é criado na inicialização do programa (MPI2 dinâmico)
- Modelo de programação usual é SPMD

Funções básicas - Inicialização

- MPI_Init(&argc, &argv)

  Inicializa uma execução MPI, é responsável por copiar o código em todos processadores
- MPI_Finalize()

  Termina uma execução MPI
- Exemplo

```
#include <stdio.h>
#include "mpi.h"
main(int argc, char** argv) {
   MPI_Init(&argc, &argv);
   ...
   MPI_Finalize();
}
```

Funções Básicas - Inicialização

- Paralelismo se da de forma implícita
- Não existem funções para criação e terminação de processos
- Identificação dos processos é fundamental para comunicação
- Communicators viabilizam a comunicação de grupo de processos e a programação modular

Funções básicas - Inicialização

- MPI_Comm_rank(comm, pid);
  - Determina o identificador do processo corrente
  - IN: comm communicator(handle)
  - OUT: pid: identificador do processo no grupo comm (int)
- MPI_Comm_size(comm, size);
  - Determina o número de processos em uma execução
  - IN: comm communicator(handle)
  - OUT: size: número de processos no grupo comm (int)

Funções básicas - Inicialização

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

main(int argc, char** argv) {
  int mypid;
  int np;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &mypid);
  MPI_Comm_size(MPI_COMM_WORLD, &np);
  MPI_Finalize();
}
```

Funções básicas - Comunicação

- Comunicação ponto-a-ponto
  - Funções MPI_Send e MPI_Recv
  - Bloqueante ou não bloqueante
- Comunicação coletiva
  - Outras funções de broadcast e concentração de informações

Funções básicas - Comunicação

- MPI_Send(buf, count, datatype, dest, tag, comm)

  IN buf: endereço do buffer de envio
  IN count: número de elementos a enviar
  IN datatype: tipo dos elementos a enviar
  IN dest: identificador do processo de destino
  IN tag: (etiqueta) da mensagem
  IN comm: communicator (handle)

Funções básicas - Comunicação

- MPI_Recv(buf, count, datatype, source, tag, comm, status)

  OUT buf: endereço do buffer de recebimento
  IN count: número de elementos a receber
  IN datatype: tipo dos elementos a receber
  IN dest: identificador do processo de origem
  IN tag: (etiqueta) da mensagem
  IN comm: communicator (handle)

Funções básicas - Comunicação

```
#include <stdio.h>
#include "mpi.h"
main(int argc, char** argv) {
  int mypid;
  int np;
  int source;
  int dest;
  int tag=50;
  char message[100];
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &mypid);
  MPI_Comm_size(MPI_COMM_WORLD, &np);
```

Funcoes básicas - Comunicação

```
if (mypid!=0) {
   sprintf(message, "greetings from %d!", mypid);
   dest=0;
   MPI_Send(message, strlen(message)+1, MPI_CHAR,
            dest, tag, MPI_COMM_WORLD);
}
else {
  for (source=1; source< np; source++) {
    MPI_Recv(message, 100, MPI_CHAR, source, tag,
             MPI_COMM_WORLD, &status);
    printf("%s\n", message);
  }
}
MPI_Finalize();
}
```

Funções básicas – sincronização

- Sincronização implícita
  - Através de comunicação bloqueante
- Sincronização explícita e global
  - Uso do isend e irecv
  - Através de barreiras
  - MPI_Barrier(comm)

    IN comm: communicator (handle)

Funções básicas - Barreira

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv) {
  int rank,size;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  if (rank == 0) {
      printf("(%d) -> Primeiro a escrever!\n",rank);
      MPI_Barrier(MPI_COMM_WORLD);
  } else {
      MPI_Barrier(MPI_COMM_WORLD);
      printf("(%d) -> Agora posso escrever!\n",rank);
  }
  MPI_Finalize();
  return 0;
}
```

MPI - Conclusões

- Existem diversas implementações (MPICH, LAM)
  - Geralmente com compilador próprio
  - Ferramenta de execução própria
- Ferramenta de depuração e editores
  - Debuggers
  - Profilers
  - Sistema hospedeiro
- É atualmente o mais utilizado em programação paralela (existem outros como o PVM) e vem diminuindo com o aumento do uso de OpenMP (memória compartilhada)
  - Aumento de máquinas *multicore*