Message Passing Interface (MPI): Introdução e Primeiros Programas

Paulo Sérgio Lopes de Souza
pssouza@icmc.usp.br

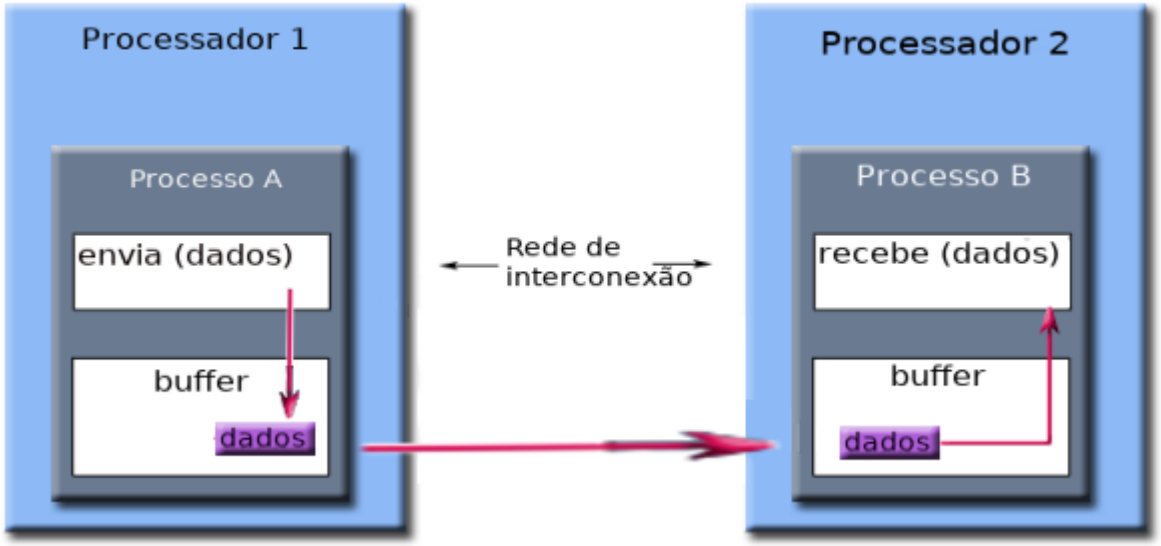
Universidade de São Paulo / ICMC / SSC – São Carlos
Laboratório de Sistemas Distribuídos e Programação Concorrente

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico
CAPES
FAPESP

O Modelo de Passagem de Mensagens

- Assumem-se alguns atributos chave
  - Paralelização explícita no código
    - Geração de processos usualmente por *fork-join*
  - Espaço de endereçamento de memória não compartilhado entre processos
    - Há apenas **memórias locais** aos processos
    - Modelo voltado máquinas MIMD com **memória distribuída** (*clusters*, Grids, ...)

- **Comunicação e sincronização** entre os processos são **associadas**
  - *send* e *receive* permitem **a troca (passagem) de mensagens** entre processos
    - ***send(void *sendbuf, int nelems, int dest)***
    - ***receive(void *recevbuf, int nelems, int source)***



Adaptado de: https://computing.llnl.gov/tutorials/mpi/

O Modelo de Passagem de Mensagens

- Há outras primitivas de comunicação, algumas de mais alto nível:
  - *Rendezvous*
  - *Comunicações coletivas*
  - *Remote Procedure Call* (RPC), *Remote Method Invocation* (RMI) e *Web Services*

- O programador deve considerar que:
  - Novamente, dados pertencem à memória local ao processo:
    - Dados devem ser divididos e atribuídos adequadamente;
  - Interações usualmente bilaterais:
    - Ambos os processos cooperam para a comunicação (*send & receive*);
  - Programador deve (tem condições de) reduzir a comunicação da aplicação
    - Otimização do desempenho;
  - Localidades espacial e temporal são determinantes para o desempenho.
  - Flexível por ser aplicado eficientemente em diferentes arquiteturas
    - Alterando-se a granularidade
  - Programas tendem a ser mais complexos
    - Quando comparados à programação via memória compartilhada

O Modelo de Passagem de Mensagens

- Programas com passagem de mensagens são considerados
  - Assíncronos porque as atividades concorrentes executam assincronamente;
  - Fracamente sincronizados porque interagem menos que as threads;

- Execução na CPU é **não determinística**

- Não há compartilhamento de memória
  - Não há regiões críticas, exclusão mútua e condições de disputa

- MPMD (*Multiple Program, Multiple Data*) ou SPMD (Single Program, Muliple Data)
  - MPMD é mais flexível e mais complexo. Pode dificultar a escalabilidade.
  - SPMD é mais simples de organizar e voltado mais ao paralelismo de dados.

- Diferentes iniciativas focam no modelo de Passagem de Mensagens
  - Estabeleceram diferentes padrões ao longo dos anos
  - Básico: linguagem estruturada (como C) e API sockets sobre TCP/IP

  - Ambientes de Passagem de Mensagens
    - Removeram complexidades da programação
    - Trouxeram problema de falta de padronização
      - P4, Parmacs, Express, Linda, *Parallel Virtual Machine* (PVM), ...
    - *Message Passing Interface* (**MPI**) tentativa de padronização

Message Passing Interface

- **MPI é um padrão da indústria**; não uma implementação
  - Estabelece como deve ser implementada a comunicação entre processos por passagem de mensagem
  - Tem especificações para rotinas de comunicação e sincronização por passagem de mensagens
  - Para C, C++ e Fortran
  - Visa portabilidade, eficiência e flexibilidade

- Breve histórico:
  - MPI 1.3 (setembro/2008) – 129 funções
  - MPI 2.2 (setembro/2009) – 300 funções
  - MPI 3 (setembro/2012) – 437 funções
  - MPI 4 (último draft de junho/2020)

- Algumas implementações do padrão MPI:
  - **OpenMPI** https://www.open-mpi.org/
  - MPICH: http://www.mpich.org/
  - Intel MPI library
  - IBM Spectrum MPI
  - HP MPI
  - SGI MPI
  - Sun MPI

Message Passing Interface

- Elementos básicos da programação C com MPI

  Include
  #include <mpi.h>

  Compilação
  mpicc -o programa programa.c

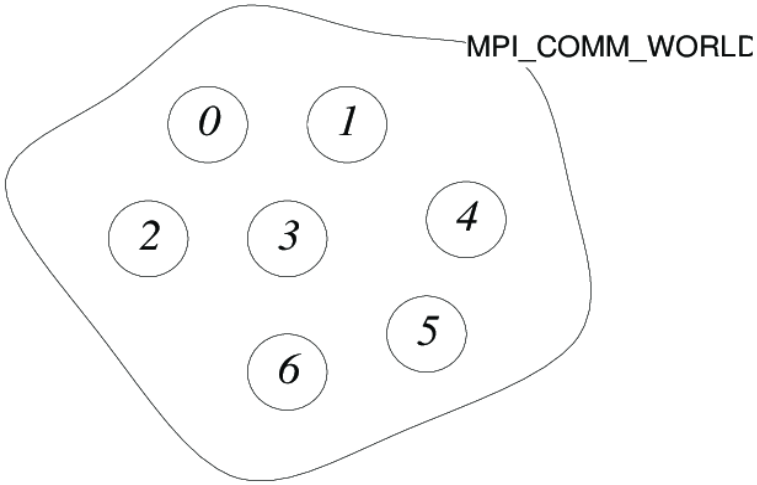
  Execução (básica)
  mpirun -np <num> executável [args]

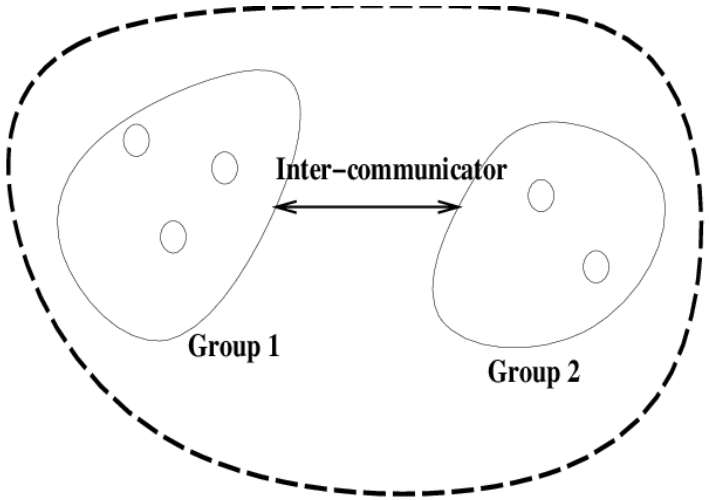
- Na execução:
  - ***-np*** especifica quantidade de processos a serem criados; cada um executa uma cópia do executável (SPMD)
  - Há limites para ***–np***: quantidade de slots disponíveis na arquitetura onde o executável será executado. ***Slots*** representam quantidade de processadores físicos disponíveis.
  - ***--use-hwthread-cpus*** permite usar também os processadores lógicos
  - ***--oversubscribe*** permite mais de um processo por ***slot***
  - ***--hostfile <hostfilename>***,especifica nós que podem ser usados para atribuir os processos gerados (mapeamento de processos em processadores).
    - O parâmetro ***hostfilename*** é um arquivo texto com endereços ou nomes dos nós (hosts/máquinas).
    - Os nós podem ser especificados sem o ***–hostfile*** (usar ***–host/-host/-H***)

https://www.open-mpi.org/doc/v4.0/man1/mpirun.1.php.

Message Passing Interface

- Contexto de comunicação e comunicadores no MPI (***MPI_COMM_WORLD***)
  - **Intracomunicadores** e **Intercomunicadores**





MacDonald et al. (2020) Fagg et al. (1997)

Message Passing Interface

- Tipos de Dados no MPI

Tipo do MPI	Tipo do C
MPI_CHAR	char
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG_INT	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wide char
MPI_PACKED	special data type for packing
MPI_BYTE	single byte value

Message Passing Interface

Primeiras funções (básicas) do MPI:

int MPI_Init(int *argc, char ***argv)

int MPI_Finalize()

int MPI_Comm_size(MPI_Comm comm, int *size)

int MPI_Comm_rank(MPI_Comm comm, int *rank)

int MPI_Send(void *bufferE, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm)

int MPI_Recv(void *bufferR, int count, MPI_Datatype datatype, int source,
int tag, MPI_Comm comm, MPI_Status *status).

MPI_ANY_SOURCE
MPI_ANY_TAG

Message Passing Interface

- **Exemplos de programa C/MPI**: *Hello world*! ☺

hello.c

há dois processos; P0 envia msg para P1, recebe a de volta (*ping-pong*) e imprime resultado no STDOUT

hellofor.c

há P processos; P0 envia uma msg para cada processo (não para ele mesmo, obviamente); recebe P-1 msgs de volta e as imprime conforme chegam

- **Exercícios solicitados**:

1) Dado um vetor vet[TAM], determinar quantos nrs maiores que vet[K] existem em vet e em quais posições eles estão, conforme especificação.

2) Faça um programa concorrente em C/MPI que implemente um *Token Ring* de processos, conforme especificação.

Referências

Rauber, T., & Rünger, G. (2013). *Parallel Programming*. Springer. Second edition. Capítulo 5.

Pacheco, P. (2011). *An introduction to parallel programming*. Elsevier. Capítulo 3.

Barlas, G. (2014). *Multicore and GPU Programming: An integrated approach*. Elsevier. Capítulo 5.

Grama, A., Kumar, V., Gupta, A., & Karypis, G. (2003). Introduction to parallel computing. Pearson Education. Capítulo 6.

Apostila de Treinamento: Introdução ao MPI (Unicamp). https://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_MPI.pdf

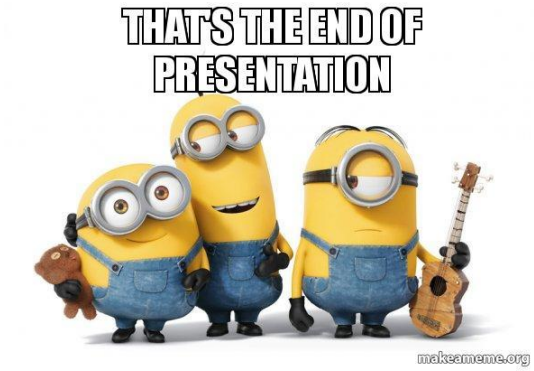
MacDonald, N; Minty, E.; Malard, J.; Harding, T.; Brown, S.; Antonioletti, M. Writing Message Passing Parallel Programs with MPI. 2020. Disponível em: https://www.researchgate.net/publication/239179288_Writing_Message_Passing_Parallel_Programs_with_MPI (último acesso em 27/10/2020)

Fagg, Graham; Dongarra, Jack; Geist, Al. Heterogeneous MPI Application Interoperation and Process Management under PVMPI. 91-98. 1997. Disponível em: https://www.researchgate.net/figure/Inter-communicator-formed-inside-a-single-MPI-COMM-WORLD_fig1_221597084 (último acesso em 27/10/2020)

Message Passing Interface (MPI): Introdução e Primeiros Programas

Paulo Sérgio Lopes de Souza
pssouza@icmc.usp.br

Universidade de São Paulo / ICMC / SSC – São Carlos
Laboratório de Sistemas Distribuídos e Programação Concorrente



CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico
CAPES
FAPESP