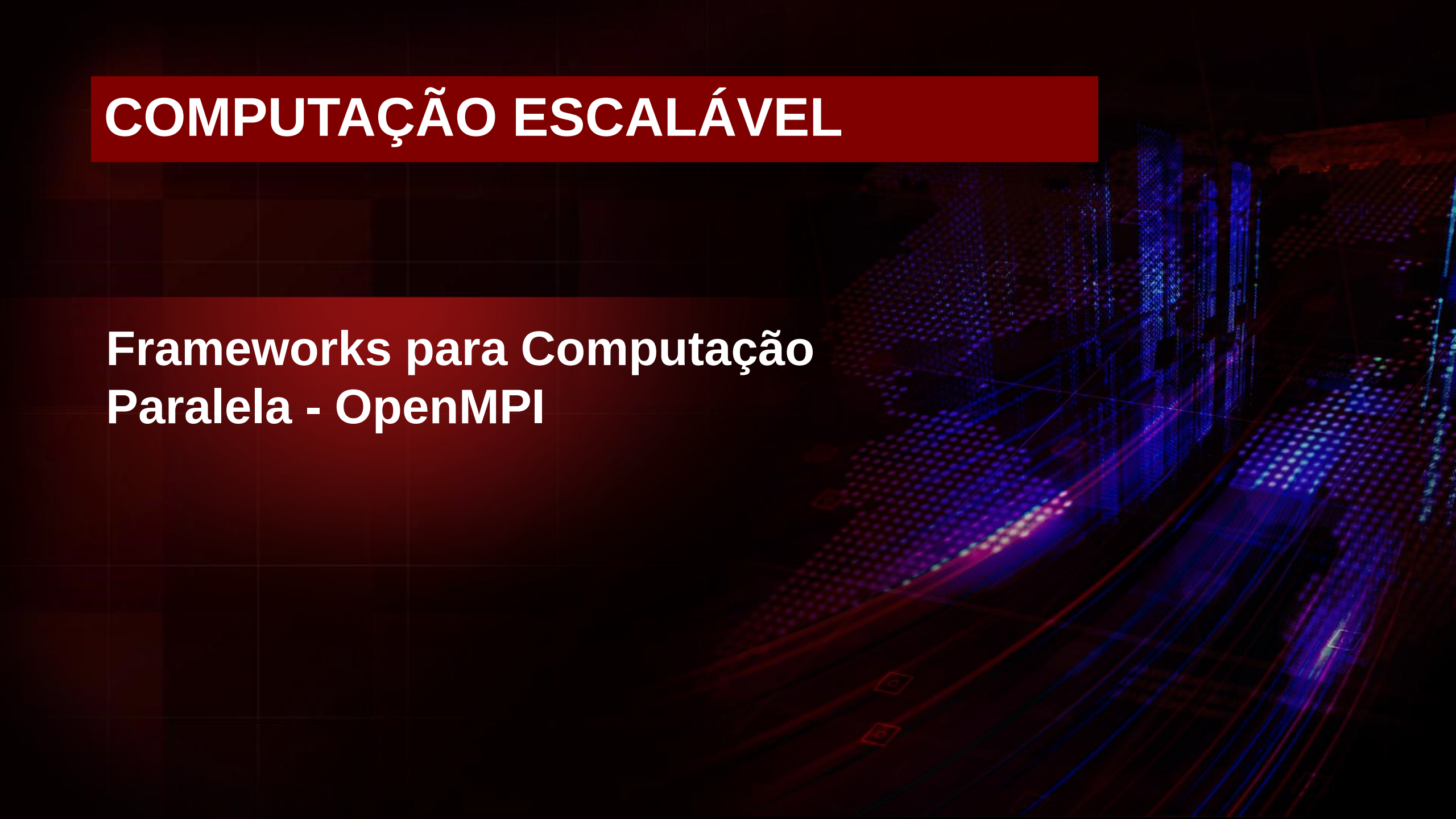
COMPUTAÇÃO ESCALÁVEL

Frameworks para Computação Paralela - OpenMPI



ROTEIRO

- O que é o OpenMPI?
- Motivação
- Características
- Conceitos Básicos
- Principais Funções

O QUE É OPENMPI?

MPI (Message Passing Interface)

O modelo de passagem de mensagens é um conjunto de processos que possuem acesso à memória local. As informações são enviadas da memória local do processo para a memória local do processo remoto

O QUE É OPENMPI?

MPI é uma biblioteca de Message Passing desenvolvida para ambientes de memória distribuída, máquinas massivamente paralelas, NOWs (network of workstations) e redes heterogêneas

Padrão criado por um consórcio entre empresas e universidades

É uma biblioteca de rotina que fornece funcionalidade básica para que os processos se comuniquem.

O paralelismo é explícito (programador é responsável pela distribuição)

MOTIVAÇÃO

- Necessidade de poder computacional
- Dificuldades na melhoria dos processadores sequenciais
- Custo x Benefício
- Dificuldades
  - Comunicação, coordenação, sincronização
  - Distribuição de trabalho

CARACTERÍSTICAS

Vantagens do MPI

- Maduro
- Portabilidade é facilidade
- Combina eficientemente com diversos hardwares
- Implementações proprietárias e públicas
- Interface com usuário
- Buffer de manipulação
- Permite abstrações de alto nível
- Desempenho

Desvantagens do MPI

- O ambiente de controle de execução depende da implementação
- Comunicação entre os hosts pode ser o gargalo da solução

CARACTERÍSTICAS

Thread Safety

- Comunicação ponto-a-ponto
- Modos de comunicação: standard, synchronous, ready, buffered
- Buffers estruturados
- Tipos de dados derivados

Comunicação Coletiva

- Nativo já incorporado, operações coletivas definidas pelo usuário
- Rotinas de movimentação de dados

Perfis

- Usuários podem interceptar chamadas MPI e chamar suas próprias ferramentas

CONCEITOS BÁSICOS

Processo

- Cada parte do programa quebrado é chamado de processo. Os processos podem ser executados em uma única máquina ou em várias

Rank

- Todo o processo tem uma identificação única atribuída pelo sistema quando o processo é inicializado. Essa identificação é contínua e representada por um número inteiro, começando de zero até N-1, cujo N é o número de processos. Cada processo tem um rank, e ele é utilizado para enviar e receber mensagens

CONCEITOS BÁSICOS

Rank

- Usado pelo desenvolvedor para especificar a fonte e o destino de mensagens
- Pode ser usado condicionalmente pela aplicação para controlar a execução do programa

```
if rank == 0
        do ...
else if rank == 1
        do …
```

CONCEITOS BÁSICOS

Grupos

- Grupo é um conjunto ordenado de N processos. Todo e qualquer grupo é associado a um comunicador muitas vezes já predefinido como "MPI_COMM_WORLD"

Comunicador

- O comunicador é um objeto local que representa o domínio (contexto) de uma comunicação (conjunto de processos que podem ser contatados). O MPI_COMM_WORLD é o comunicador predefinido que inclui todos os processos definidos pelo usuário numa aplicação MPI
- As chamadas de MPI_Send e MPI_Recv correspondentes devem possuir o mesmo comunicador

CONCEITOS BÁSICOS

Application Buffer

- É o endereço de memória, gerenciado pela aplicação, que armazena um dado que o processo necessita enviar ou receber

System Buffer

- É um endereço de memória reservado pelo sistema para armazenar mensagens.

Count

- Indica o número de elementos de dados de um tipo, para ser enviado

CONCEITOS BÁSICOS

Type

- O MPI pré define tipos de dados elementares

Tag

- Inteiro (não negativo). Utilizado para identificar unicamente uma mensagem. As chamadas de MPI_Send e MPI_Recv correspondentes devem possuir a mesmo tag

Status

- Para uma operação de receive, indica a fonte da mensagem e a tag da mensagem. Em C, um ponteiro para uma estrutura pré-definida (MPI_STATUS). Adicionalmente, o número de bytes é obtido do status, via rotina MPI_Get_Count

PRINCIPAIS FUNÇÕES

- Em geral são utilizadas:
  - MPI_Init
  - MPI_Finalize
  - MPI_Comm_size
  - MPI_Comm_rank
  - MPI_Get_processor_name

PRINCIPAIS FUNÇÕES

MPI_Init

- Inicializa um processo MPI
- Deve ser a primeira rotina a ser chamada por cada processo, pois estabelece o ambiente necessário para executar o MPI
- Sincroniza todos os processos na inicialização de uma aplicação MPI
- Utilizada para passar argumentos de linha de comando para todos os processos

MPI_Finalize

- MPI_Finalize é chamada para encerrar o MPI. Ela deve ser a última função a ser chamada. É usada para liberar memória. Não existem argumentos

PRINCIPAIS FUNÇÕES

Exemplo com MPI_Init e MPI_Finalize

Código C:

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

  int main(int argc, char **argv)
{
  int  rc;
  rc = MPI_Init(&argc,&argv);
  if (rc == MPI_SUCCESS) {
    printf ("MPI iniciou
corretamente.\n");
    }

  MPI_Finalize();
  return 0;
  }
```

Saída com quatro processadores:

```
MPI iniciou corretamente.
MPI iniciou corretamente.
MPI iniciou corretamente.
MPI iniciou corretamente.
```

PRINCIPAIS FUNÇÕES

MPI_Comm_size

- Determina o número de processos no grupo, associado com o comunicador
- Usado com o comunicador MPI_COMM_WORLD para determinar o número de processos utilizados pela aplicação
- Obtém o comunicador como seu primeiro argumento, e o endereço de uma variável inteira é usada para retornar o número de processos

MPI_Comm_rank

- Identifica um processo MPI dentro de um determinado grupo comunicador. Retorna sempre um valor inteiro entre 0 e n-1, cujo n é o número de processos

PRINCIPAIS FUNÇÕES

Exemplo com MPI_Comm_size e MPI_Comm_rank

Código C:

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

int main(int argc, char **argv)
 {
   int  numtasks, rank, rc;
   rc = MPI_Init(&argc,&argv);
   if (rc == MPI_SUCCESS){
      MPI_Comm_size (
                      MPI_COMM_WORLD,
                      &numtasks);
      MPI_Comm_rank(
                     MPI_COMM_WORLD,
                     &rank);
      printf ("Sou o processo %d de %d\n",
rank, numtasks);
     }
   MPI_Finalize(); return 0;
   }
```

Saída com quatro processadores:

```
Sou o processo 0 de 4
Sou o processo 3 de 4
Sou o processo 1 de 4
Sou o processo 2 de 4
```

PRINCIPAIS FUNÇÕES

MPI_Get_processor_name

- Retorna o nome do nó, cujo processo individual está executando
- Usa um argumento para guardar o nome da máquina e outro para o tamanho do nome

PRINCIPAIS FUNÇÕES

Exemplo com MPI_Get_processor_name

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

Int main( int argc, char * argv[  ] )
{
      int  processId;      /* rank dos processos */
      int  noProcesses;    /* Número de processos */
      int  nameSize;       /* Tamanho do nome */
      char computerName[MPI_MAX_PROCESSOR_NAME];

 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &noProcesses);
 MPI_Comm_rank(MPI_COMM_WORLD, &processId);
 MPI_Get_processor_name(computerName, &nameSize);

fprintf(stderr,"Hello from process %d on %s\n", processId, computerName);

 MPI_Finalize( );
return 0;    }
```

BIBLIOGRAFIA

1. https://hpc-tutorials.llnl.gov/mpi/
2. Introduction to Parallel Computing, Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar - 2ª ed., Addison Wesley
3. Programação Paralela e Distribuída - https://www.dcc.fc.up.pt/~ricroc/aulas/1011/ppd/apontamentos/mpi.pdf