Arquiteturas Paralelas e Distribuídas

Ferramentas para programação paralela: bibliotecas MPI

Eduardo Furlan Miranda

Baseado em: IGNÁCIO, A. A. V.; FERREIRA FILHO, V. J. M. MPI: uma ferramenta para implementação paralela. 2002. DOI 10.1590/S0101-74382002000100007.

Message Passing Interface (MPI)

- Padrão de interface para a troca de mensagens em máquinas paralelas com memória distribuída
- Desenvolvido por várias instituições, principalmente dos EUA e Europa, universidades, laboratórios, e governos
- Uma aplicação é constituída por um ou mais processos que se comunicam, acionando-se funções para o envio e recebimento de mensagens entre os processos
- O número de processos no MPI é normalmente fixo
- A comunicação pode ser ponto a ponto, ou um grupo de processos pode invocar operações coletivas de comunicação para executar operações globais

- Suporta comunicação assíncrona e programação modular, através de mecanismos de comunicadores (communicator)
  - permitem definir módulos que encapsulem estruturas de comunicação interna

- Os algoritmos que criam um processo para cada processador podem ser implementados, diretamente, utilizando-se comunicação ponto a ponto ou coletivas

- Os algoritmos que implementam a criação de tarefas dinâmicas ou que garantem a execução concorrente de muitas tarefas, num único processador, precisam de um refinamento nas implementações com o MPI

MPI - Básico

A sintaxe dos comandos varia conforme a linguagem de programação

import mpi4py	Inicia uma execução MPI
MPI.Finalize()	Finaliza a execução
comm.Get_size()	Determina o número de processos
comm.Get_rank()	Determina a identificação de processos
comm.send(data, dest=1, tag=11)	Envia a mensagens
comm.recv(source=0, tag=11)	Receber mensagens

- Os procedimentos geralmente possuem um manipulador de comunicação como argumento
  - identifica o grupo de processos e o contexto das operações
  - proporciona o mecanismo para identificar um subconjunto de processos, durante o desenvolvimento de programas modulares, assegurando que as mensagens, planejadas para diferentes propósitos, não sejam confundidas

Exemplo - send, recv

```
from mpi4py import MPI

# Inicialização do MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Processo 0 envia uma mensagem para o processo 1
if rank == 0:
    data = {'a': 1, 'b': 2}
    comm.send(data, dest=1, tag=11)
    print("Processo 0 enviou dados para o processo 1")
elif rank == 1:
    data = comm.recv(source=0, tag=11)
    print("Processo 1 recebeu dados do processo 0:", data)

# **Barreira:** Sincronização de todos os processos
comm.Barrier()
print(f"Processo {rank} passou pela barreira")

# **Finalização do MPI**
MPI.Finalize()
```

Exemplo

```
from mpi4py import MPI    # A inicialização do MPI é feita automaticamente
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
if rank == 0:    # Processo 0 envia dados para todos os outros processos
    data = np.array([1.0, 2.0, 3.0])
else:
    data = np.empty(3, dtype=np.float64)  # Cria um array vazio para receber
comm.Bcast(data, root=0)
print(f"Processo {rank} recebeu (bcast) os dados: {data}")
# **Gather:** Todos os processos enviam dados para o processo 0
send_data = np.array([rank * 1.0, rank * 2.0, rank * 3.0])  # a serem enviados
recv_data = None
if rank == 0:
    recv_data = np.empty([size, 3], dtype=np.float64)  # recebe os dados
comm.Gather(send_data, recv_data, root=0)
if rank == 0:
    print("Processo 0 recebeu (gather) os dados de todos os processos:")
    print(recv_data)
MPI.Finalize()    # Encerra e não se pode acessar outras funções MPI
```

Exemplo

```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
    int world_size;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    printf("Olá do processo %d de um total de %d!\n", world_rank, world_size);
    MPI_Finalize();
    return 0;
}
```

- mpicc -o meu_programa mpi_programa.c
- mpirun -np 4 ./meu_programa

MPI

- A programação de troca de mensagens não é determinística ou seja, a chegada das mensagens enviadas por dois processos A e B ao processo C não é definida
- Não garante que uma mensagem, enviada de um processo A e de um processo B, chegue na ordem que foi enviada
- O programador deve assegurar a ordem, se for importante
- RECV pode permitir o recebimento de mensagens vindas de um único processo específico, ou de qualquer processo
- “tag” : mecanismo adicional para distinguir as mensagens

Operações Globais

- Operações coordenadas envolvendo múltiplos processos
  - P. ex., todos os processos podem precisar transpor uma matriz distribuída ou somar um conjunto de números distribuídos em cada um dos processos
- As operações podem ser implementadas por um programador, utilizando-se funções de recebimento e envio de mensagens
- Por comodidade e para permitir implementações otimizadas, o MPI também fornece uma serie de funções especializadas de comunicação que executam operações comuns desse tipo

- Barreira (Barrier): sincroniza todos os processos de um grupo
  - Nenhum processo pode realizar qualquer instrução, até que todos tenham passado por essa barreira
    - Maneira simples de separar as duas etapas da execução para assegurar que as mensagens geradas não se misturem
      - Em alguns casos a necessidade de barreiras explícitas pode ser evitada pelo uso de tag, de origem e/ou contextos específicos
- Difusão (Broadcast): envia dado de um processo a todos os demais processos
  - Dispersão de dados do tipo um para todos, no qual um único processo origem envia um dado para todos os outros processos e cada processo recebe esse dado

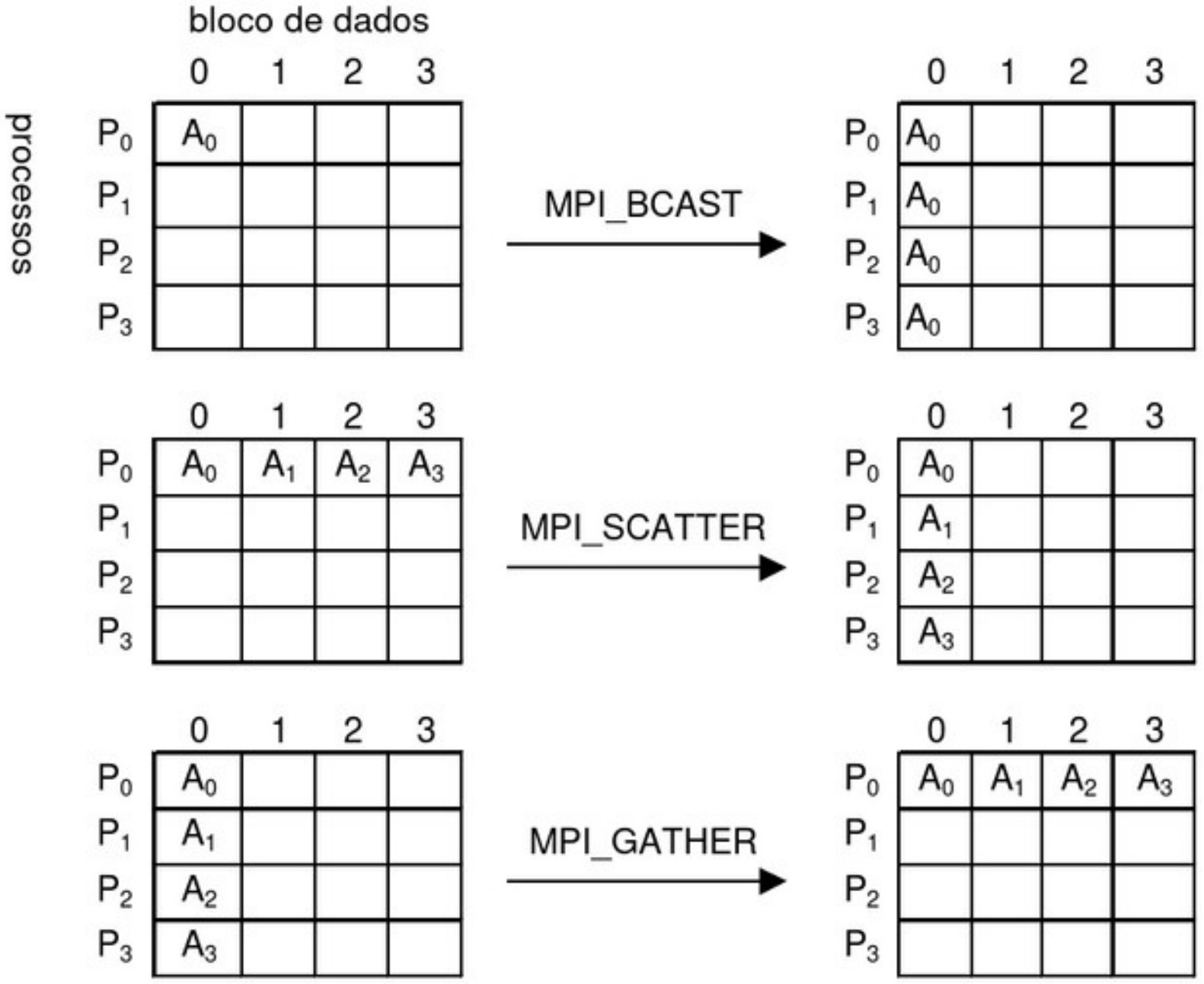
- Juntar (Gather): junta dados de todos os processos em um único processador
  - Todos os processos, incluindo a origem, enviam dados localizados na origem
  - Esse processo coloca os dados em locais contíguos e não opostos, com dados do processo “i”, precedendo os dados do processo “i+1”

- Espalhar (Scatters): distribui um conjunto de dados de um processo para todos os processos (reverso da GATHER)
  - enquanto na BCAST, todo processo recebe o mesmo valor do processo de origem, no SCATTER, todo processo recebe um valor diferente

- comm.Bcast( ), comm.Gather, e comm.Scatter(): rotinas coletivas de movimentação de dados, nas quais todos os processos interagem com origens distintas para espalhar, juntar e distribuir os dados, respectivamente

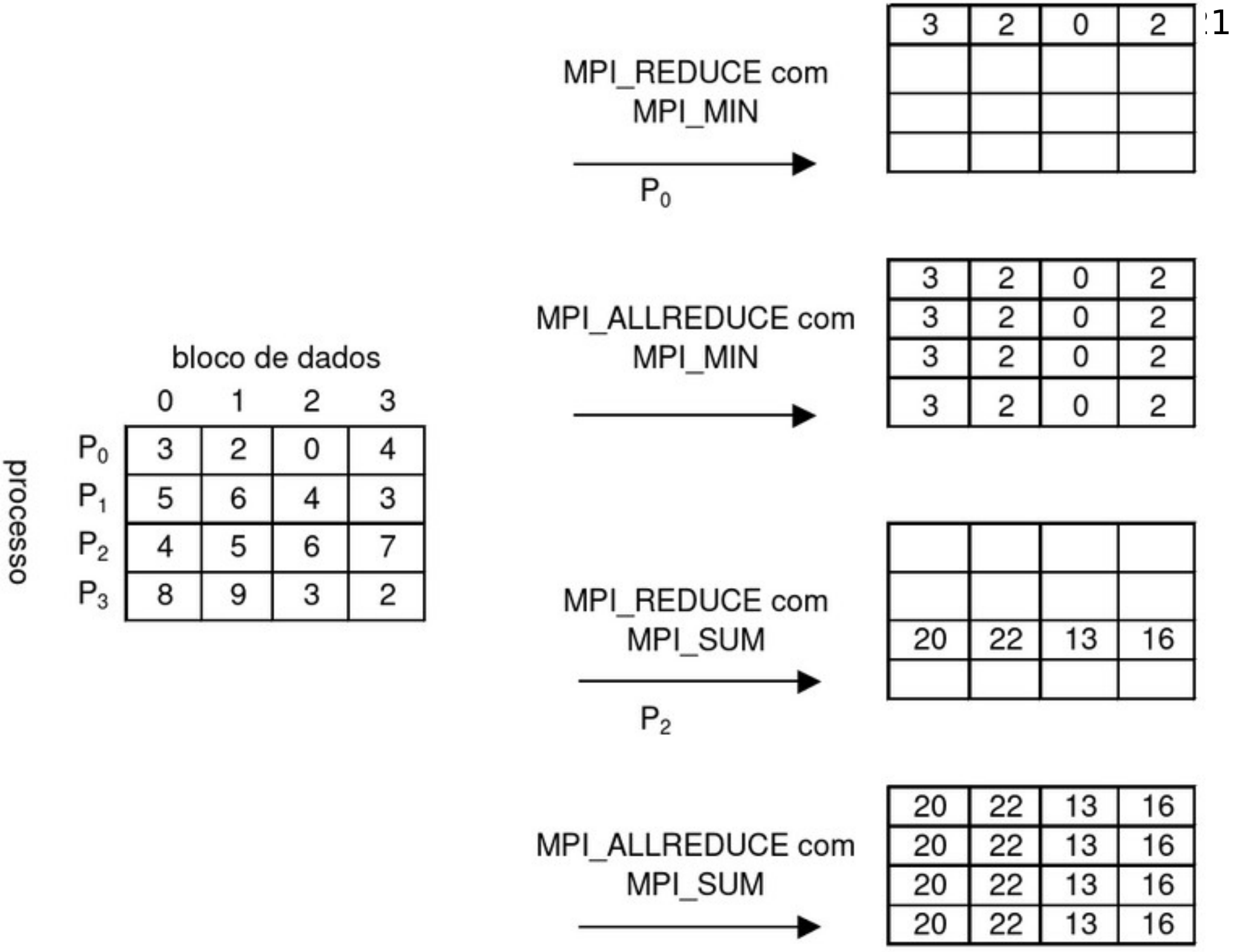
- Em cada caso, os primeiros três argumentos especificam o local e o tipo de dados a serem comunicados e o número de elementos a serem enviados para cada destino

- Outros argumentos especificam o local, o tipo de resultado e o número de elementos a serem recebidos de cada origem


bloco de dados
0 1 2 3
processos
P0
P1
P2
P3
A0
MPI_BCAST
A0
A1
A2
A3
MPI_SCATTER
MPI_GATHER

- Operação de redução (Reduction operations): soma, multiplicação, etc., de dados distribuídos

- comm.Reduce() e comm.Allreduce() implementam operações de redução
  - Eles combinam valores fornecidos no bufer de entrada de cada processo, usando operações especificas, e retorna o valor combinado, ou para o bufer de saída de um único processo da origem (REDUCE) ou para os bufer de saída de todos os processo (ALLREDUCE)
  - Essas operações incluem o máximo, o mínimo (MPI.MAX, MPI.MIN), soma, produto (MPI.SUM, MPI.PROD) e as operações lógicas


bloco de dados
0 1 2 3
processo
P0 3 2 0 4
P1 5 6 4 3
P2 4 5 6 7
P3 8 9 3 2
MPI_REDUCE com MPI_MIN
P0
3 2 0 2
MPI_ALLREDUCE com MPI_MIN
3 2 0 2
3 2 0 2
3 2 0 2
MPI_REDUCE com MPI_SUM
P2
20 22 13 16
MPI_ALLREDUCE com MPI_SUM
20 22 13 16
20 22 13 16
20 22 13 16
20 22 13 16

- Outros recursos
  - a possibilidade de comunicação assíncrona, quando um cálculo acessa elementos de uma estrutura de dados compartilhados em um modo não estruturado
  - a composição modular usada quando são implementados programas grandes de sistemas complexos onde módulos são implementados independentemente
  - o uso de ferramentas que ajudam a melhorar a performance dos algoritmos, investigando o ambiente dentro do qual estão sendo executados
  - o uso de diferentes tipos de dados, usados por exemplo para fazer a conversão entre diferentes representações de dados em ambientes heterogêneos

MPI-2

- incorporação de gerenciamento dinâmico de processos
- manipulação de Threads
- tem-se a possibilidade de restringir o acesso à comunicação de algumas Threads, como por exemplo, apenas uma Thread (a principal) pode executar as funções do MPI

Compilação e Execução

- Os métodos de compilação e execução dependem da implementação utilizada, mas, de forma genérica, os principais passos para compilar e executar um programa em MPI são:

- Utilizar compiladores que reconheçam automaticamente os procedimentos MPI, ou incluir manualmente as bibliotecas MPI no processo de compilação

- Verificar quais são os nós disponíveis no ambiente paralelo

- Fixar os parâmetros para o ambiente paralelo: por exemplo, escolher os nós a serem usados e o protocolo de comunicação

- Executar o programa (que deverá ser executado em cada um dos nós escolhidos)

Implementações de MPI

- MPICH: Implementação de alta performance e código aberto, mantida pelo Argonne National Laboratory. Foco em escalabilidade e portabilidade para diversas plataformas. http://www.mpich.org/

- Open MPI: Projeto de código aberto unificado, resultado da fusão de várias implementações MPI. Ampla comunidade e suporte, oferece recursos avançados e otimizações de desempenho. https://www.open-mpi.org/

Outras Implementações

- Intel MPI Library: Otimizada para arquiteturas Intel, oferece alto desempenho e integração com ferramentas de desenvolvimento Intel.

- IBM Spectrum MPI: Implementação da IBM para seus sistemas de alto desempenho, com foco em escalabilidade e confiabilidade.

Escolhendo a Implementação MPI

- A escolha da implementação MPI depende dos requisitos do seu projeto:
  - Desempenho: Open MPI e Intel MPI Library são frequentemente consideradas as mais rápidas, especialmente em hardware Intel.
  - Portabilidade: MPICH e Open MPI oferecem ampla compatibilidade com diferentes sistemas operacionais e arquiteturas.
  - Suporte: Open MPI possui uma grande comunidade e suporte comercial disponível.
  - Recursos Avançados: Algumas implementações oferecem recursos específicos, como suporte a RDMA ou integração com ferramentas de profiling.

- Recomendação:
  - Para a maioria dos casos, Open MPI é uma excelente opção, oferecendo um equilíbrio entre desempenho, portabilidade e suporte.
  - Se você precisa de desempenho máximo em hardware Intel, considere a Intel MPI Library