Insper

# SuperComputação

## Aula 18 – Comunicação coletiva e assíncrona

2019 – Engenharia

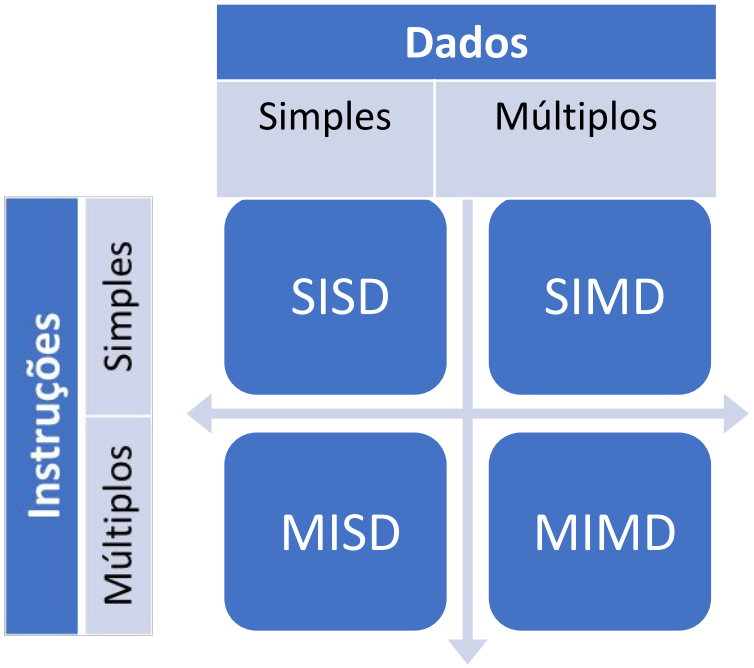
Igor Montagner, Luciano Soares <igorsm1@insper.edu.br>

# Objetivos

- Comunicação coletiva entre processos
- Usar a API do boost::mpi para comunicação coletiva

# Taxonomia de Flynn

- Descreve como um sitema paralelo funciona em dois eixos: dados e instruções



# Arquiteturas de comunicação

- Mestre/Escravo
  - Mestre distribui tarefas
  - Escravos executam as tarefas e reportam resultados

- Cliente/Servidor
  - Cliente requisita tarefas/dados
  - Servidor distribui tarefas/dados

# Mensagens coletivas

- Comunicação entre processos
  - Todos para todos
  - Todos para um
  - Um para todos

# Barrier

Todos os processos param até chegar naquele ponto. Igual às barreiras do OpenMP.

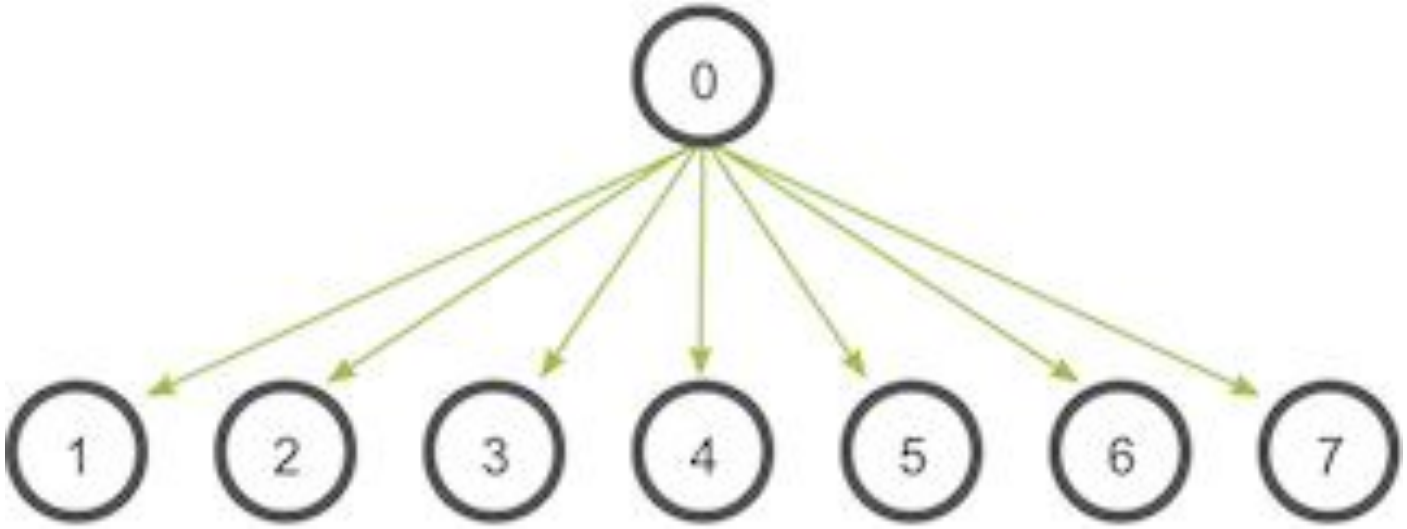
*communicator::barrier();*

Cuidado: todos os processos precisam alcançar o código, senão o programa inteiro trava!

# Broadcast

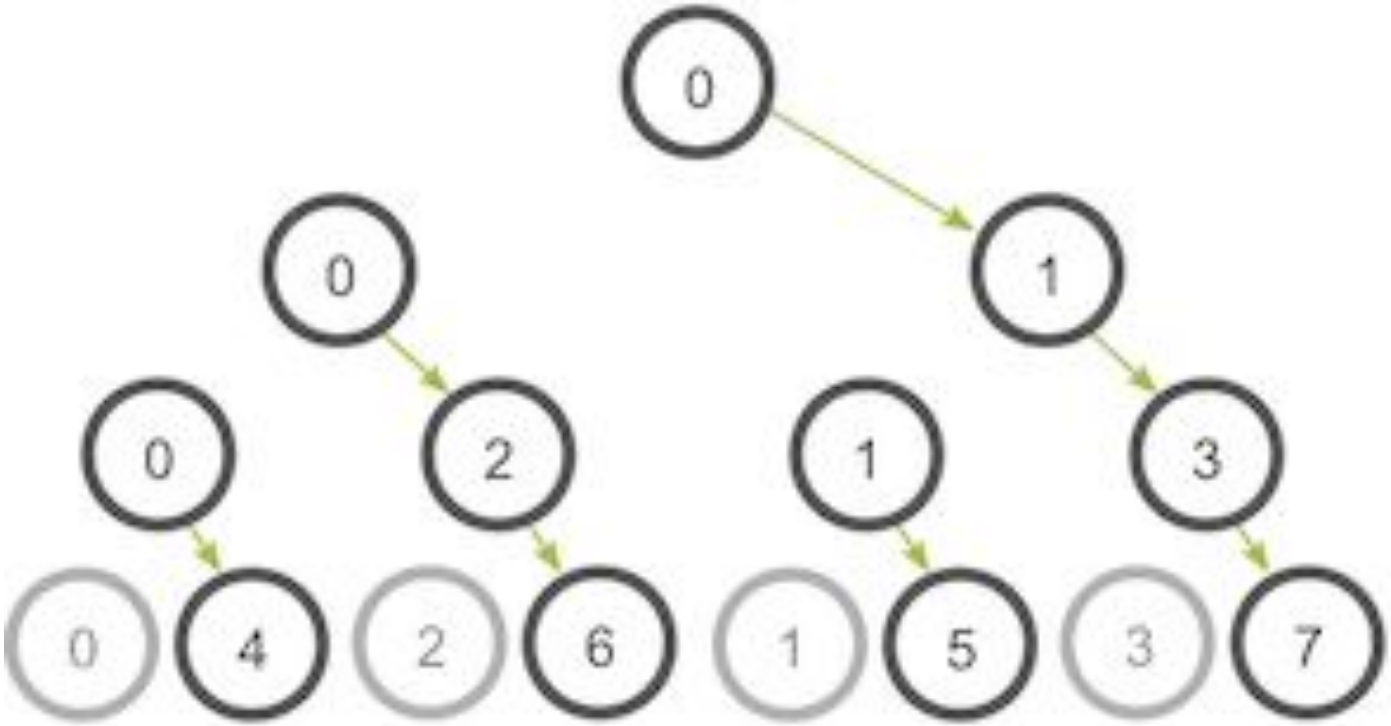
Replica um dado para todos os processos.

*boost::mpi::broadcast(world, data, root_rank);*



# Broadcast - vantagem

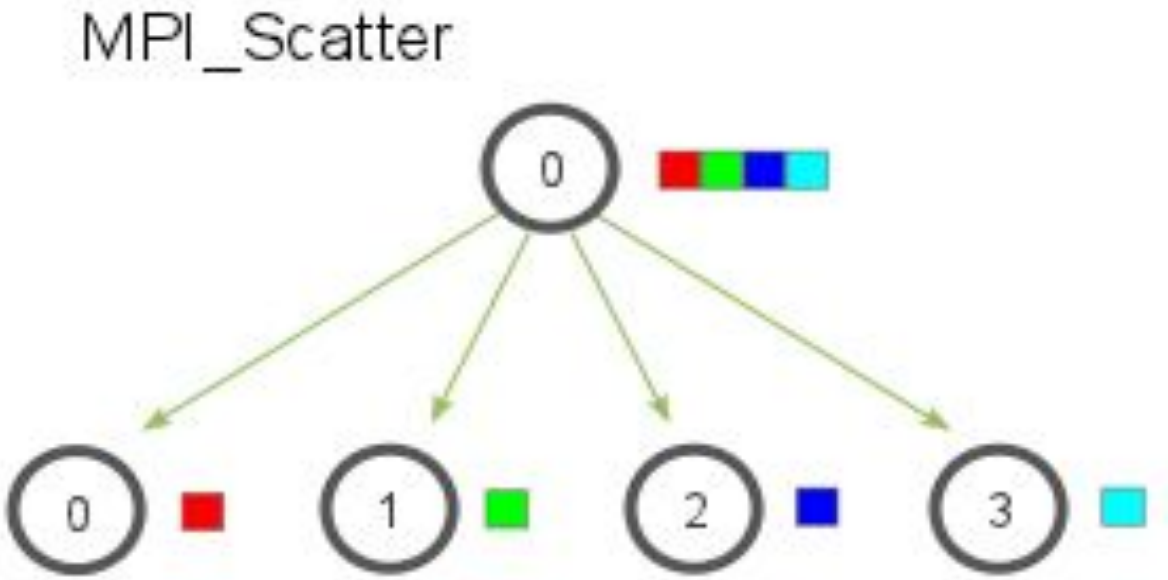
Usa algoritmo inteligente de distribuição



# Scatter

Divide dados entre os processos, de modo que cada um pegue um pedaço de igual tamanho.

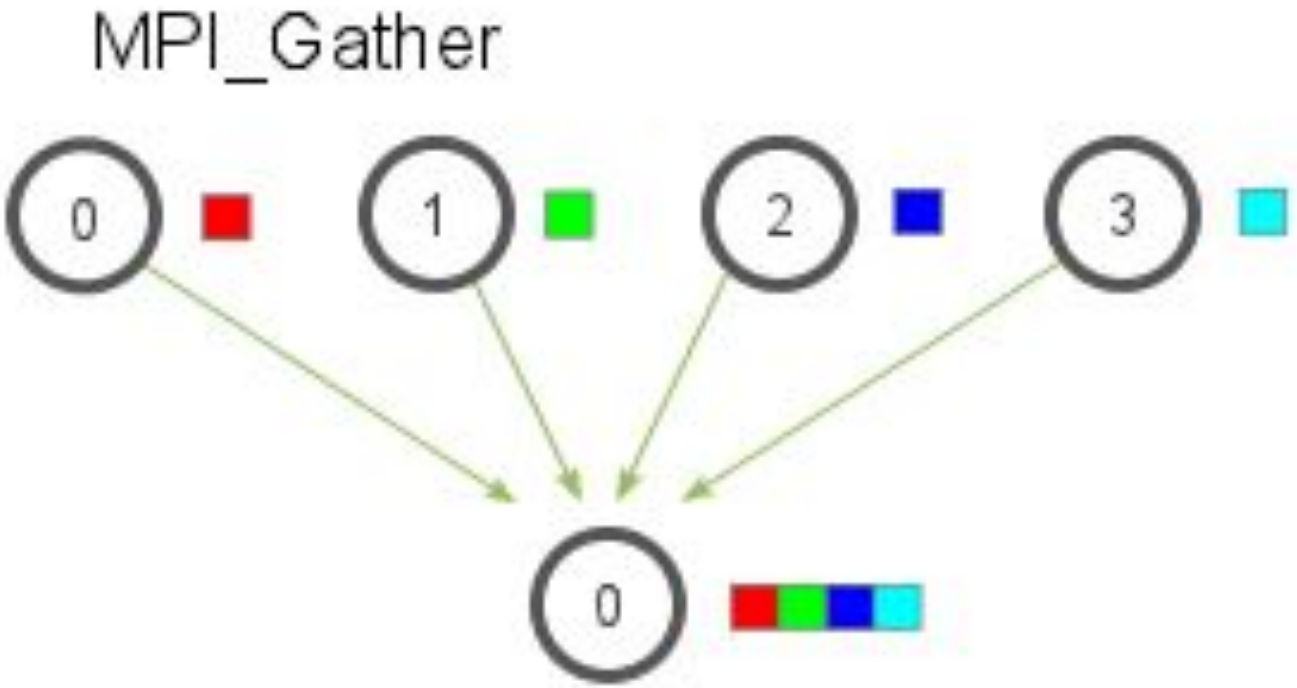
*void scatter(const communicator & comm, const std::vector< T > & in_values, T &out_value, int root);*



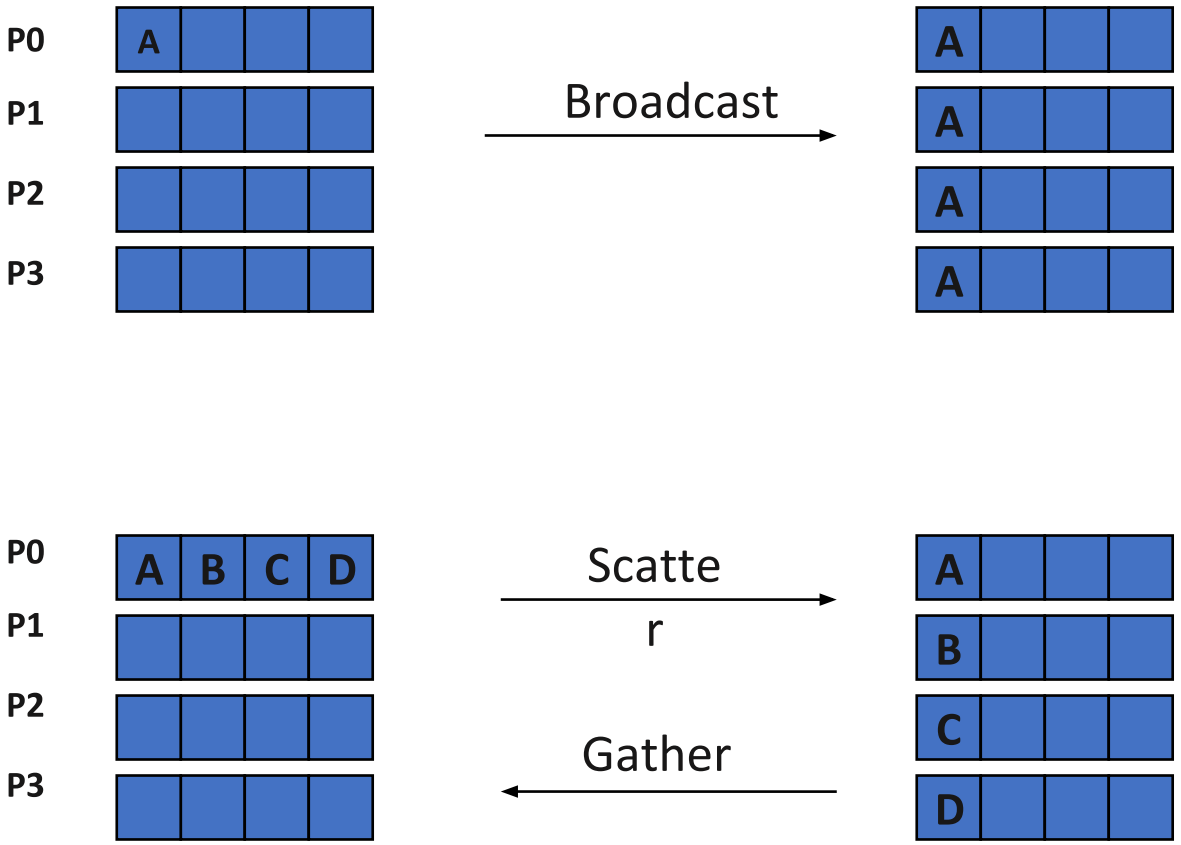
# Gather

Recolhe informações de vários processos e junta no processo de rank *root*

*void gather(const communicator & comm, const T & in_value, std::vector< T > & out_values, int root);*



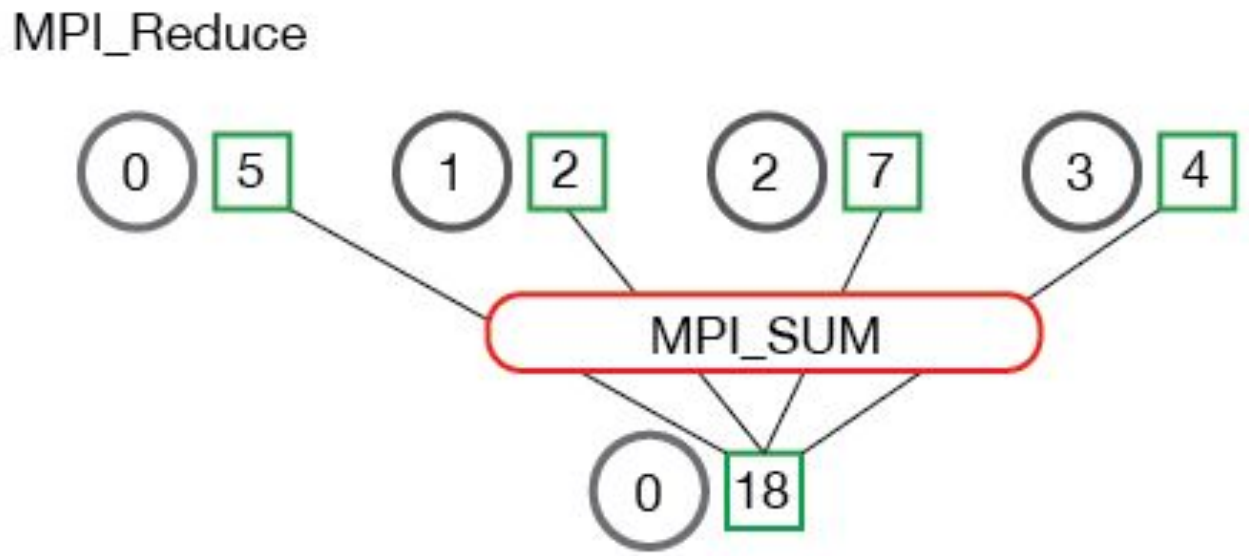
# Broadcast vs Scatter vs Gather



# Reduce

Junta informações e aplica função, igual ao OpenMP

*void reduce(const communicator & comm, const T & in_value, T & out_value, Op op, int root);*



# Reduce - operações

| C Constant | Boost.MPI Equivalent |
|---|---|
| MPI_BAND | bitwise_and |
| MPI_BOR | bitwise_or |
| MPI_BXOR | bitwise_xor |
| MPI_LAND | std::logical_and |
| MPI_LOR | std::logical_or |
| MPI_LXOR | logical_xor |
| MPI_MAX | maximum |
| MPI_MAXLOC | unsupported |
| MPI_MIN | minimum |
| MPI_MINLOC | unsupported |
| MPI_Op_create | used internally by Boost.MPI |
| MPI_Op_free | used internally by Boost.MPI |
| MPI_PROD | std::multiplies |
| MPI_SUM | std::plus |

# Atividade prática — parte 1

- Aula 18 no Github

# Mensagens assíncronas

- Programa declara **intenção** de enviar ou receber dados
- Envio/recebimento de fato ocorre em paralelo **enquanto o resto do programa roda**
- Funções para checar se a comunicação terminou e/ou esperar pelo término

# Mensagens assíncronas - Vantagens

- Receber mensagens fora de ordem!
  - Mestre divide o trabalho em N-1 processos
  - Recebe os resultados na ordem em que são terminados
- Receber e enviar dados grandes
  - Comunicação ocorre em paralelo, é possível realizar outras tarefas enquanto isto.

# Mensagens assíncronas- Desvantagens

- Código mais complexo
- Só se transforma em ganhos de desempenho se for bem usado.

# Mensagens assíncronas- API

- isend, irecv são versões assíncronas que recebem os mesmos parâmetros
- *boost::mpi::request* representa uma mensagem enviada/recebida
  - *wait* espera até que a comunicação seja concluída
  - *test* verifica se a comunicação foi concluída, mas não bloqueia

# Mensagens assíncronas- API

- Família de funções wait_* espera por vários requests ao mesmo tempo:
  - wait_all: todos requests terminaram
  - wait_any: exatamente um request terminou
  - wait_some: um ou mais terminaram

# Atividade prática — parte 2

- Aula 18 no Github

# Referências


- Livros:
  - STERLING, T.; ANDERSON, M. High Performance Computing: Modern Systems and Practices. 1. ed. Morgan Kaufmann Publishers, 2017.
- Internet:
  - https://www.boost.org/doc/libs/1_67_0/doc/html/mpi/tutorial.html
  - https://theboostcpplibraries.com/boost.mpi-collective-data-exchange
  - https://computing.llnl.gov/tutorials/mpi_advanced/DavidCronkSlides.pdf
  - http://mpitutorial.com/tutorials/

Insper

www.insper.edu.br