Computação de alta performance com MPI em Python

Renan S. Silva

Quem

- Renan S. Silva
- Formado em Ciencia da Computacao;
- Mestrando em Computação Aplicada;
- github, freenode: `h3nnn4n`
- reddit: `bacon_unleashed`

O que

- MPI = *Message Passing Interface*;
- Eh um padrão *de facto* de paralelismo;
- Várias implementações existem: MPICH, Open MPI, Intel MPI;
- Utilizados em todos os computadores do top 500, AFAIK.
- Poderoso em modelos de memória distribuída;

MPI

- Um processo eh lancado para cada node;
- Processos podem ser remotos e possuem espaços de endereçamento diferentes;
- Dados sao enviados explicitamente através de passagem de mensagens;
- Pode ser síncrono ou assíncrono;
- Processos por ser organizados em comunicadores e mensagens em tags;

Porque

- Python: Grande comunidade. Excelentes libs. Prototipagem rápida. Interface com outras linguagens (C, C++, Julia);
- Tipagem dinâmica facilita a vida :D
  - Extração automática de parâmetros;
  - Não precisa saber tamanho da mensagem *a priori*;
- GLI (Global Interpreter Lock);
- Alguns problemas sao intrinsecamente complexos e precisam de muito poder computacional para resolver;

mpi4py

- `pip install mpi4py`
- `mpiexec -n 4 python codigo_supremo.py`
- Inicialização: `mpi_init`, `COMM_WORLD`;
  - `COMM_WORLD.size`
  - `COMM_WORLD.rank`
- Funções em caps enviam *Buffer-Like objects;*
- Funções em minúsculas enviam *python objects*;
- “Toda” operação funciona em “par”;
- *Hello World.*

Algumas formas de comunicação

- *Point to Point*;
- *Broadcast*;
- *Scatter and Gatter*;
- *Allgather*;
- AlltoAll;
- Reduce;

➔ Todas são bloqueantes;
➔ Maiúsculas para *buffer objects*;
➔ Minúsculas para *python objects*;

Comunicações *Point to Point*

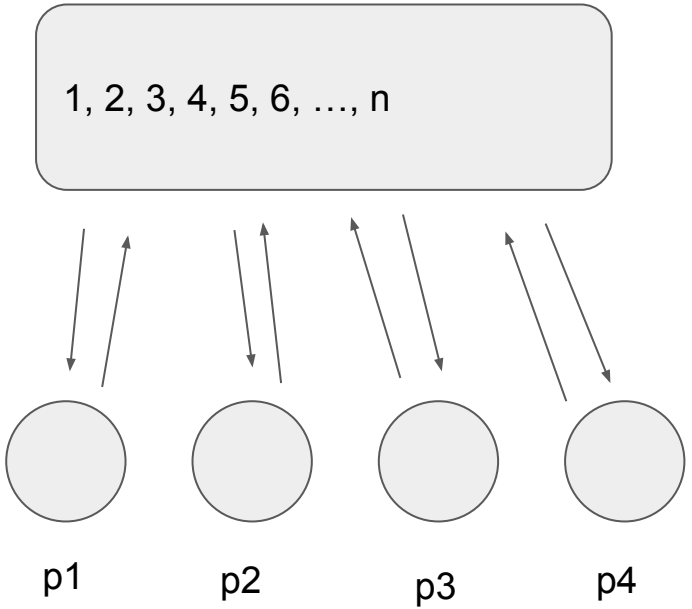
- Par de `send` e `recv`;
  - Envia uma mensagem entre dois nós;
- Funciona aos pares;
- Parâmetros:
  - `data`
  - `tag*`
  - `source*`
  - `dest*`
  - `status*`

  `* = named`

- Exemplos: 2_ping_pong, 2_ring_pong

Pool

- Um *master* com uma pool de tarefas;
- Um ou mais *slaves* que executam tarefas;
- Facil de implementar com `send` e `recv`;
- *Throughput* do *master* pode ser *bottleneck*;
  - Solução: Mais de um *Master*;
- *Loading balancing* implícito;
  - Bom para nodes heterogêneos;
  - Bom para tarefas com pesos diferentes;
- Exemplo: 3_prime_counter



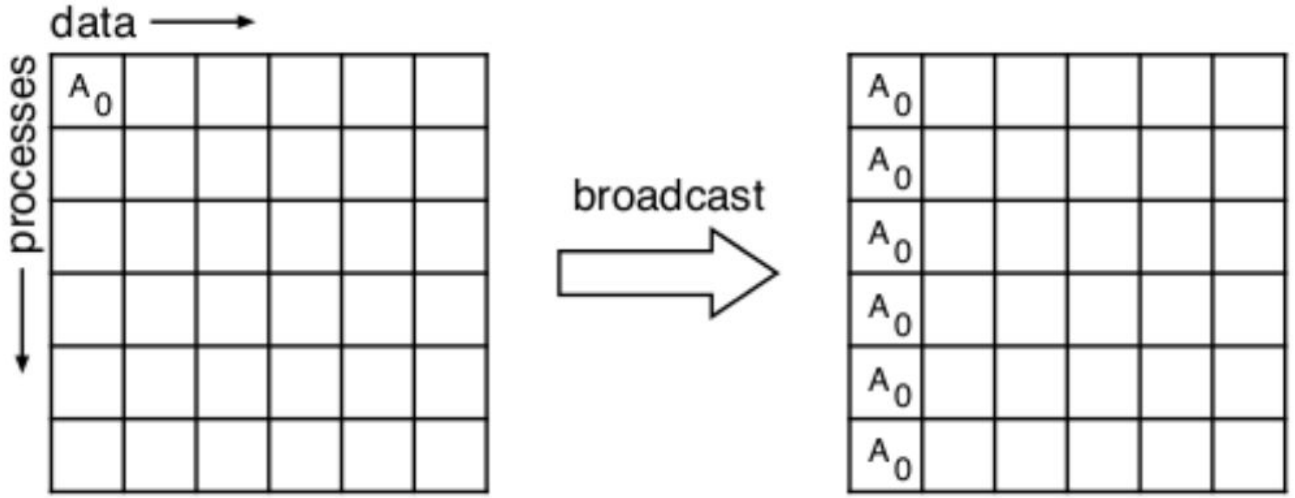
Comunicação em *Broadcast*

- `bcast` para *python objects*;
  - `Bcast` para *Buffer-Like objects*;
- Deve ser chamada em todos os nós do comunicador;
- Parâmetros:
  - `data`
  - `root*`

    `* = named`

- Exemplo: 1_hello_world_black_sheep

Comunicação em *Broadcast*



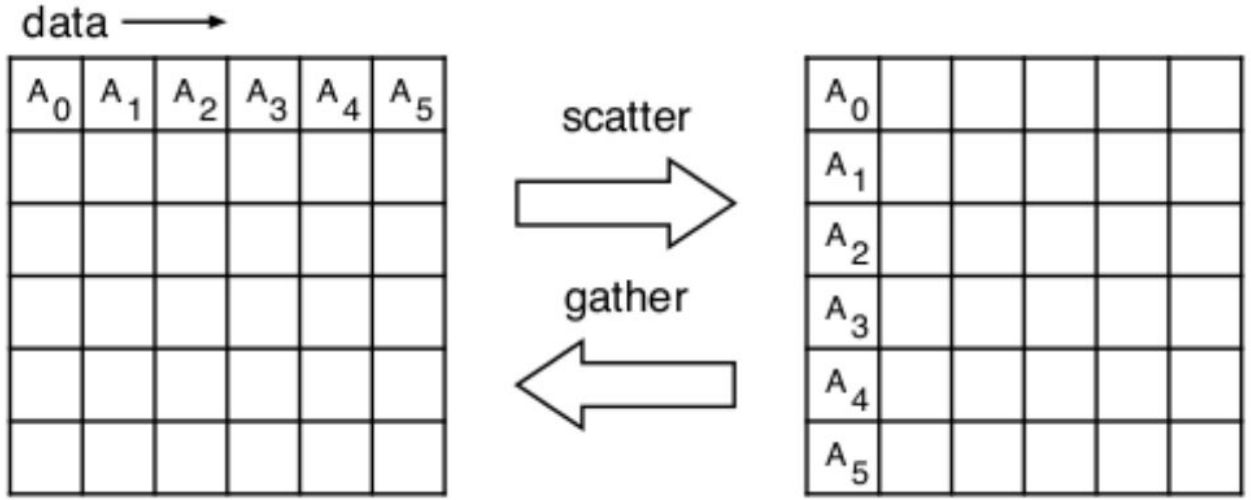
Scatter e Gather

- `scatter` e `gather` para *python objects*;
  - `Scatter` e `Gather` para *Buffer-Like objects*;
- É bloqueante;
- Deve ser chamada em todos os nós do comunicador;
- Parâmetros:
  - `data`
  - `root*`

    `* = named`

- Exemplo: 5_mandelbrot_scatter_gather, 6_std_dev, 5_fuckometro,

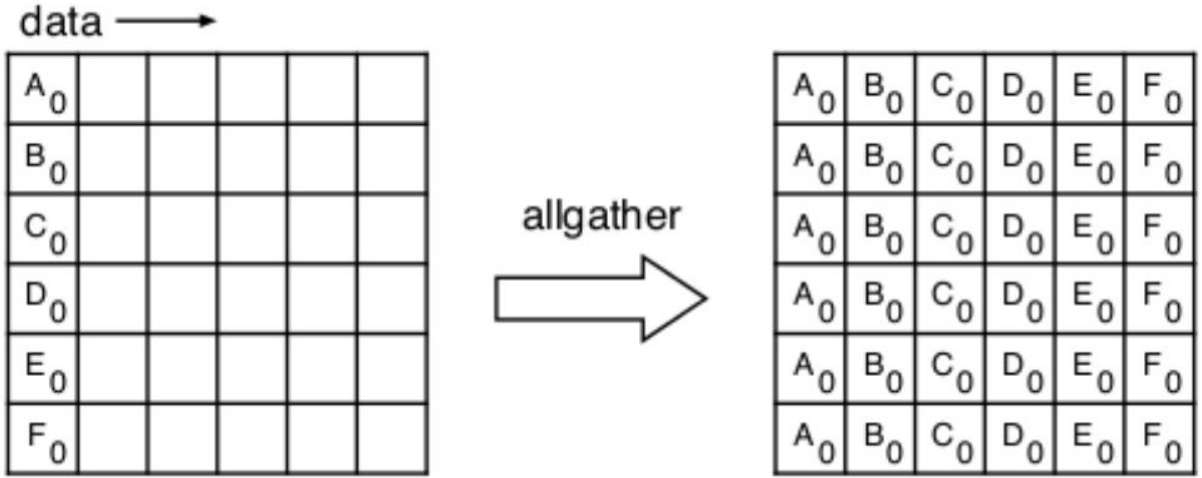
Scatter e Gather



Allgather

- `allgather` para *python objects*;
  - `Scatter` e `Gather` para *Buffer-Like objects*;
- É bloqueante;
- Deve ser chamada em todos os nós do comunicador;
- Parâmetros:
  - `data`
- Exemplo: 4_matrix_mul, 7_orbit

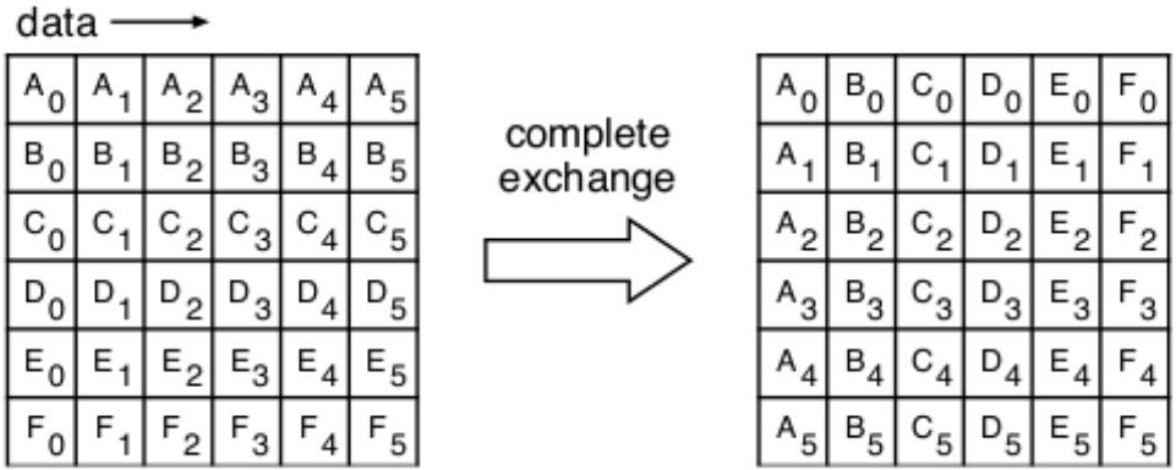
Allgather



Alltoall

- `alltoall` para *python objects*;
  - `Alltoall` para *Buffer-Like objects*;
- É bloqueante;
- Deve ser chamada em todos os nós do comunicador;
- Parâmetros:
  - `data`
- Exemplo: 8_exemplo_lixo

Alltoall



Reduce e Allreduce

- `reduce` e `allreduce` para *python objects*;
- É bloqueante;
- Deve ser chamada em todos os nós do comunicador;
- Parâmetros:
  - `data`
  - `op = MPI.MAX, MPI.MIN, MPI.SUM, MPI.PROD… (named)`
  - `root = 0 (named, reduce only)`
- Exemplo: 9_mean_std

Coisas que eu não mostrei

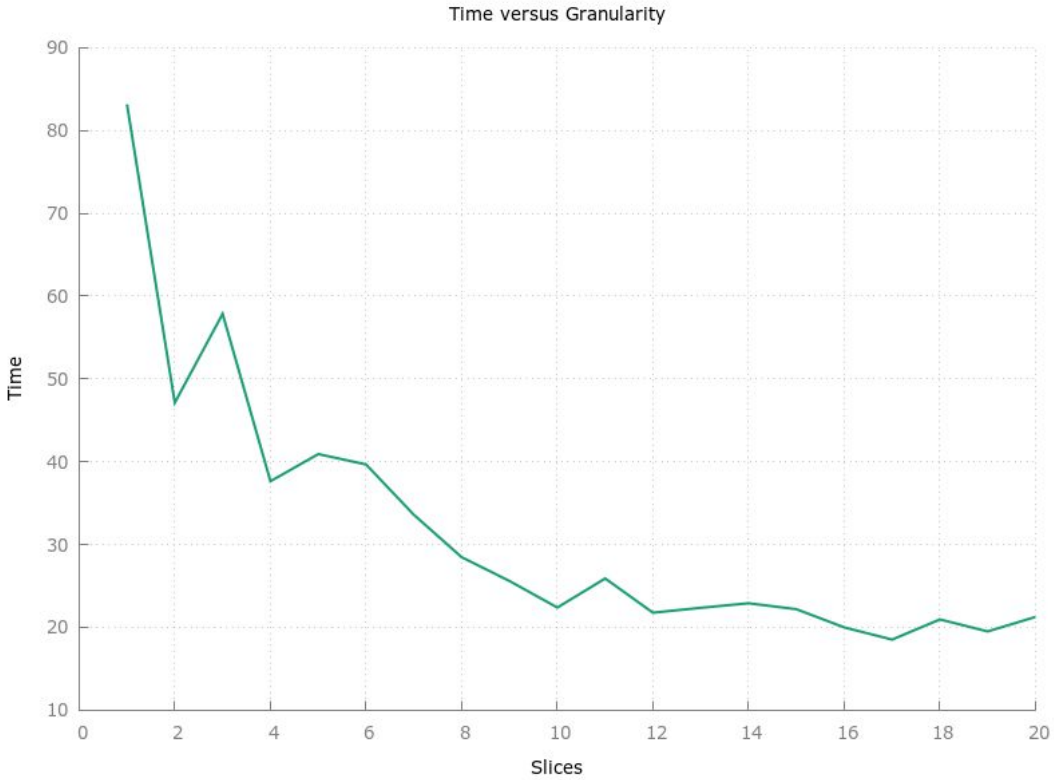
- Chamadas não bloqueantes;
- MPI Threads;
- Memória compartilhada;
- ???

Um pouquinho de teoria

Granularidade e Overhead de Comunicação

- Granularidade refere-se a quanto uma tarefa é dividida;
  - Para 4 processos, dividir uma imagem em quatro partes possui a menor granularidade. Dividir a imagem em pixels possui a maior granularidade;
- Normalmente maior granularidade implica em um maior *overhead* de comunicação;
  - Maior granularidade ajuda a evitar que um *node* fique esperando pelo outro;
- *Mandelbrot reloaded*

Granularidade e Overhead de Comunicação



Obrigado