

# Coisa é o Cálculo Paralelo e o High Performance Computing (HPC)

Leonardo De Carlo (For a complete Tutorial:  
<https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial>)

CopeLab: AI and Complexity, Universidade Lusófona

18 de março de 2025

# Cálculo de $\pi$ (Método de Monte Carlo)

- Inscreva um círculo de raio  $r$  (área  $\pi r^2$ ) dentro de um quadrado de lado  $2r$  (área  $4r^2$ ).
- A razão entre a área do círculo e a do quadrado é:

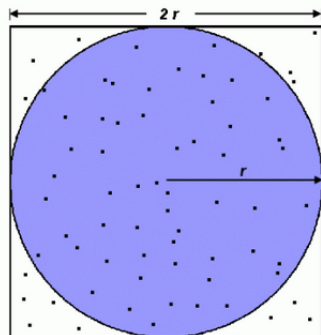
$$\frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

- Gerando aleatoriamente  $N$  pontos dentro do quadrado, aproximadamente:

$$N \times \frac{\pi}{4}$$

desses pontos ( $M$ ) devem cair dentro do círculo.

- Assim,  $\pi$  pode ser aproximado por:



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

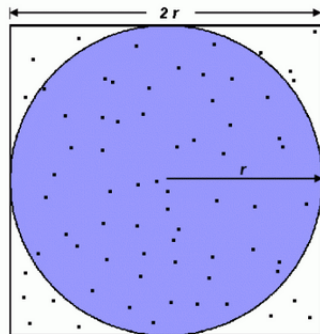
# Pseudocódigo Serial

## Pseudocódigo Serial:

```
npoints = 10000
circle_count = 0

do j = 1, npoints
  generate 2 random numbers between 0 and 1
  xcoordinate = random1
  ycoordinate = random2
  if (xcoordinate, ycoordinate) inside circ
    circle_count = circle_count + 1
  end do
```

```
PI = 4.0*circle_count/npoints
```



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

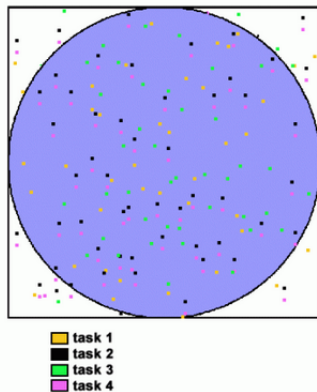
# Paralelização do Algoritmo

## Extensão para Paralelismo:

- A contagem de pontos (loop) é dividida entre os *workers*, cada um processando uma parte da tarefa.
- Cada *worker* calcula quantos pontos caem no círculo dentro do seu subconjunto.
- O *master* recebe os resultados de todos os *workers* e computa  $\pi$ .

## Observações:

- O cálculo de cada *worker* é independente, ou seja, não precisa saber o que acontece com os outros. Em geral, isso não ocorre, pois muitas computações exigem comunicação entre *workers*.
- O cálculo no passo de tempo  $n$  não depende dos passos anteriores. Isso é essencial para que a paralelização seja possível.



Calculating pi in parallel

# Pseudocódigo Paralelo (Visão Geral)

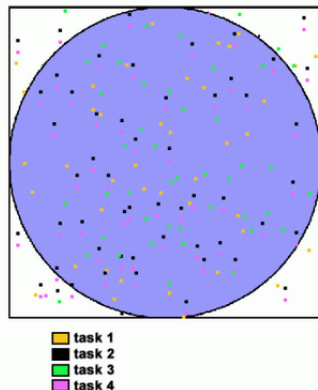
## Pseudocódigo Paralelo:

```
npoints = 10000
circle_count = 0
p = number_of_tasks
num = npoints / p

find out if I am MASTER or WORKER

do j = 1, num
  generate random x, y in [0,1]
  if inside_circle(x,y)
    circle_count = circle_count + 1
  end do

if I am MASTER
  receive circle_counts from WORKERS
  compute  $PI = 4.0 * (total\_circle\_count) / npoints$ 
else
  send circle_count to MASTER
end if
```



Calculating pi in parallel

# Paralelização em Algoritmos Dependentes do Tempo

Muitas vezes, é possível reestruturar um algoritmo dependente do tempo de forma que a paralelização seja viável.

**Exemplo: cálculo dos primeiros 10.000 termos da sequência de Fibonacci** (0,1,1,2,3,5,8,13,21,...) usando a fórmula:

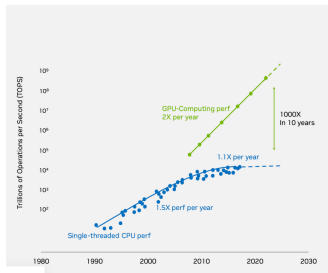
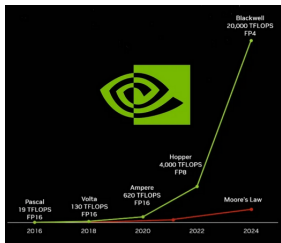
$$F(n) = F(n-1) + F(n-2)$$

O cálculo do valor  $F(n)$  depende dos valores de  $F(n-1)$  e  $F(n-2)$ , que devem ser computados primeiro. Um algoritmo paralelo para resolver esse problema é usar a **fórmula de Binet**:

$$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}} = \frac{\varphi^n - (-\varphi)^{-n}}{2\varphi - 1}, \quad \varphi = \frac{1 + \sqrt{5}}{2} \approx 1.6180339887 \dots$$

# High Performance Computing(HPC)

O cálculo paralelo está vivendo um momento de destaque com as últimas gerações de Graphic Processing Units (GPUs): uma capacidade de cálculo surpreendente no momento em que a Lei de Moore está chegando ao fim.



Aliado à datastorage com I/O de alto desempenho: massivo investimento da União Europeia (UE) em centros de HPC, [EuroHPC](#) (Resta provar que não estamos so enriquecendo os americanos: devido ao intenso I/O das GPUs, as vantagens dos clusters de GPUs(respeito CPUs) precisam ser demonstradas na maioria das situações (e.g. em simulações físicas)).

Essa nova tecnologia é a base para a explosão da popularização da inteligência artificial (AI). Outra aplicação de destaque é o uso de gêmeos digitais e simulações para a indústria.