

Introdução à Programação Paralela Prática - MPI

Carlos Renato de Souza

29-31/08/2023

Quem sou eu?

Carlos Renato

- **Área de trabalho:**
 - Métodos Numéricos / Modelagem Numérica / Processamento de Alto Desempenho no CPTEC/INPE
- **Doutorando** em Computação Aplicada:
 - Processamento de Alto Desempenho pelo **INPE** (2022-2025)
- Cálculo Numérico / Métodos Numéricos em C na **UNESP (FEG)**
- **Mestrado** em Computação Aplicada:
 - Processamento de Alto Desempenho pelo **INPE** (2016-2018)
- **Graduação** em Computação Científica - **UNITAU**

carlos.souza@inpe.br

carlos.renato.souza@gmail.com

Agenda

1. Introdução à programação paralela
2. MPI
3. Estrutura de um programa paralelo
4. Principais funções MPI
5. Exemplos iniciais
6. Análise desempenho
7. A equação do calor de Poisson/Laplace
8. Modelagem do problema
9. Solução numérica sequencial
10. Solução numérica paralela MPI + Análise desempenho
11. Introdução ao MPI - Shared Memory
12. Solução numérica paralela MPI-SHM + Análise desempenho

1. Introdução à programação paralela

- Curso básico e prático para programação científica paralela
- Exemplos práticos de física aplicada que exige paralelismo
- Previsão de tempo!!! CPTEC/INPE!
- **Programação paralela?**

1. Introdução à programação paralela

- Curso básico e prático para programação científica paralela
- Exemplos práticos de física aplicada que exige paralelismo
- Previsão de tempo!!! CPTEC/INPE!
- **Programação paralela?**
 - HPC, PAD, etc...
 - OpenMP - memória compartilhada
 - MPI - memória distribuída
 - Aceleradores - FPGAs, GPGPUs,
 - OpenMP 4.5/
 - OpenACC

1. Introdução à programação paralela

- Curso básico e prático para programação científica paralela
- Exemplos práticos de física aplicada que exige paralelismo

- Previsão de tempo!!! CPTEC/INPE!

Como pensar em paralelo?

- **Programação paralela?**
 - HPC, PAD, etc...
 - OpenMP - memória compartilhada
 - MPI - memória distribuída
 - Aceleradores - FPGAs, GPGPUs,
 - OpenMP 4.5/
 - OpenACC

1. Introdução à programação paralela

- Programação paralela?
 - Construir um muro muito longo!



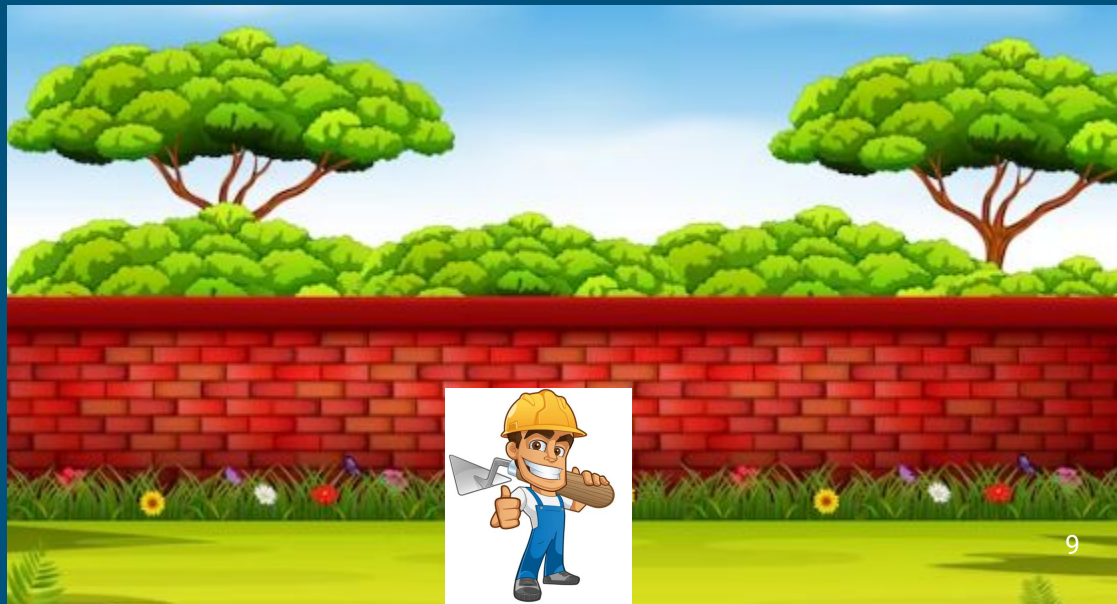
1. Introdução à programação paralela

- Programação paralela?
 - Construir um muro muito longo!
 - 100km de comprimento!



1. Introdução à programação paralela

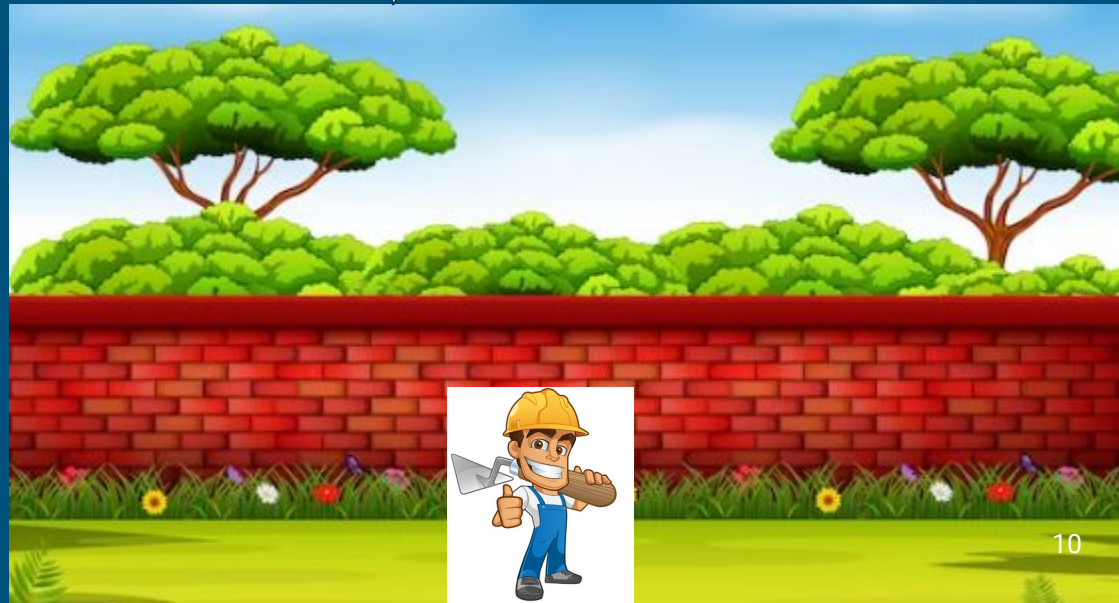
- Programação paralela?
 - Construir um muro muito longo!
 - 100km de comprimento!
 - 1 pedreiro disponível...



1. Introdução à programação paralela

- Programação paralela?
 - Construir um muro muito longo!
 - 100km de comprimento!
 - 1 pedreiro disponível...
 - Terminará o muro: 1000 dias

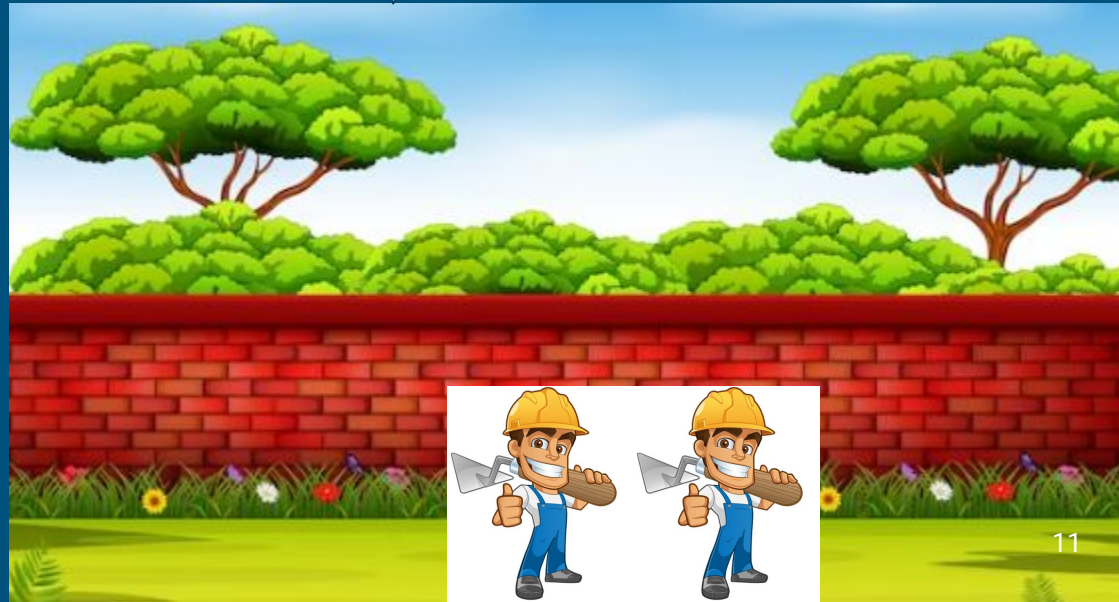
100m/dia



1. Introdução à programação paralela

- Programação paralela?
 - Construir um muro muito longo!
 - 100km de comprimento!
 - 2 pedreiros disponíveis...
 - Cada pedreiro: 50km de muro
 - Terminará o muro: 500 dias

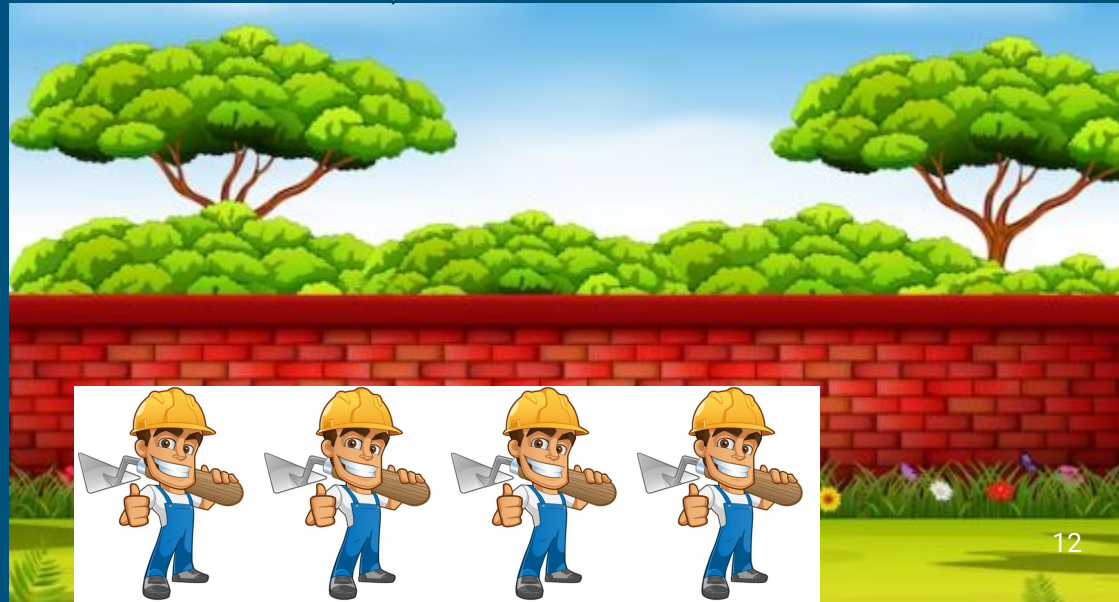
100m/dia



1. Introdução à programação paralela

- Programação paralela?
 - Construir um muro muito longo!
 - 100km de comprimento!
 - 4 pedreiros disponíveis...
 - Cada pedreiro: 25km de muro
 - Terminará o muro: 250 dias

100m/dia

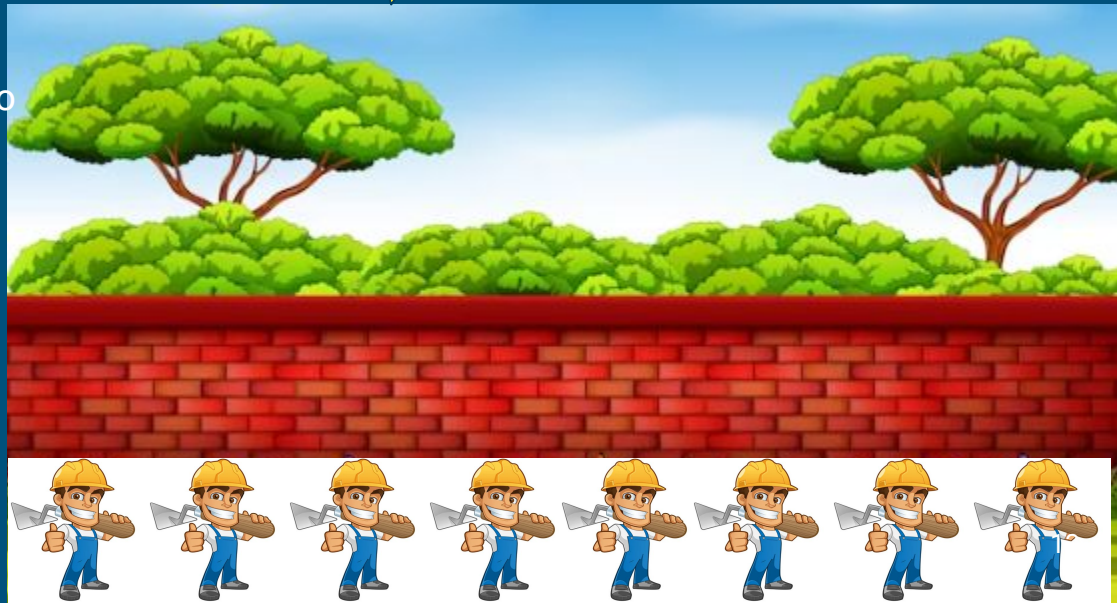


1. Introdução à programação paralela

- Programação paralela?

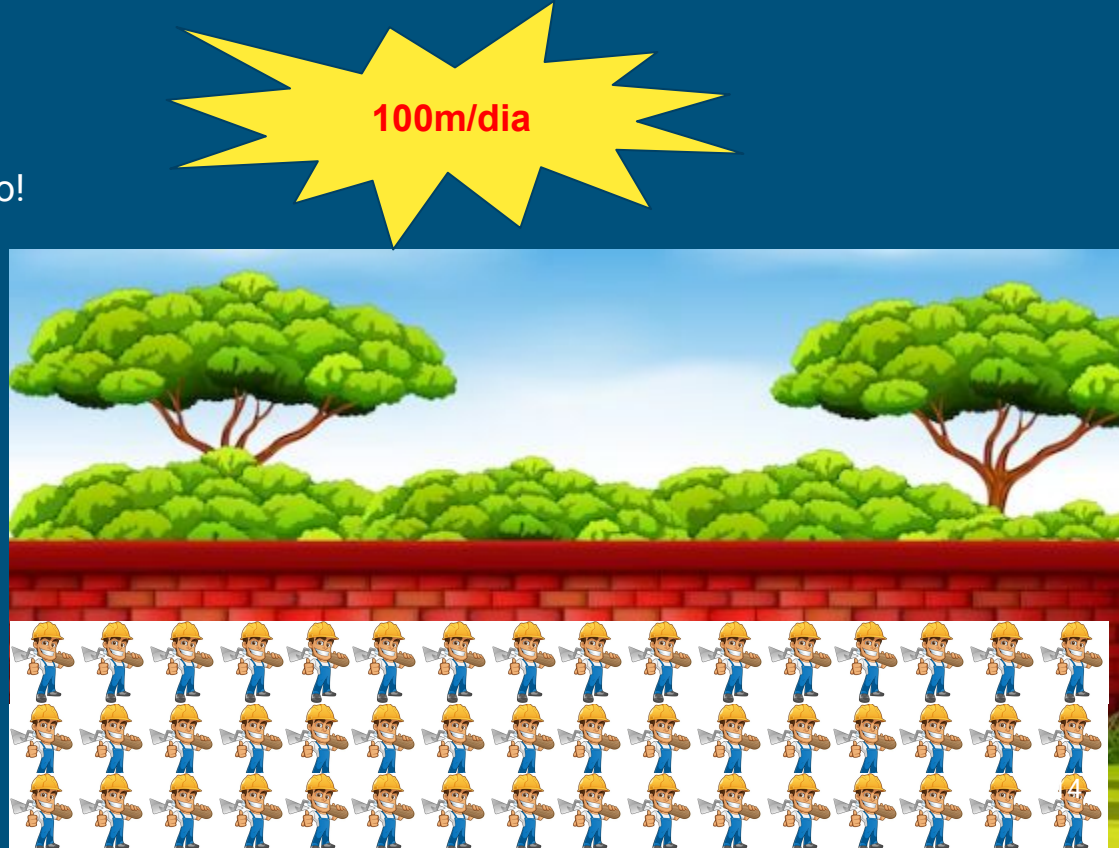
- Construir um muro muito longo!
- 100km de comprimento!
- 8 pedreiros disponíveis...
- Cada pedreiro: 12,5km de muro
- Terminará o muro: 125 dias

100m/dia



1. Introdução à programação paralela

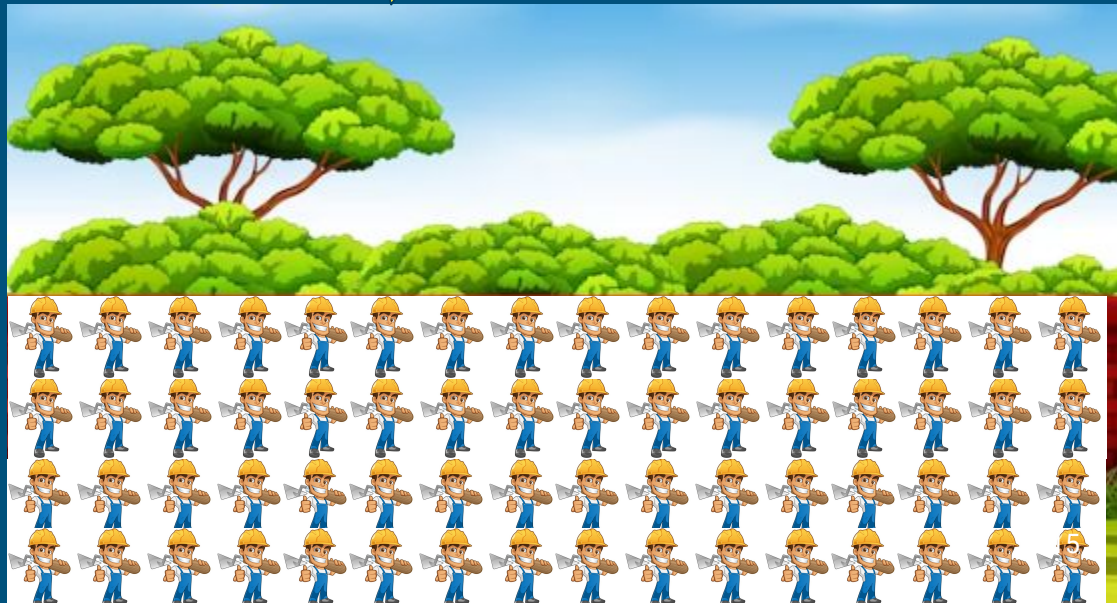
- Programação paralela?
 - Construir um muro muito longo!
 - 100km de comprimento!
 - **100** pedreiros disponíveis...
 - Cada pedreiro: **1**km de muro
 - Terminará o muro: **10** dias



1. Introdução à programação paralela

- Programação paralela?
 - Construir um muro muito longo!
 - 100km de comprimento!
 - **1.000** pedreiros disponíveis...
 - Cada pedreiro: **100**m de muro
 - Terminará o muro: **1** dia

100m/dia

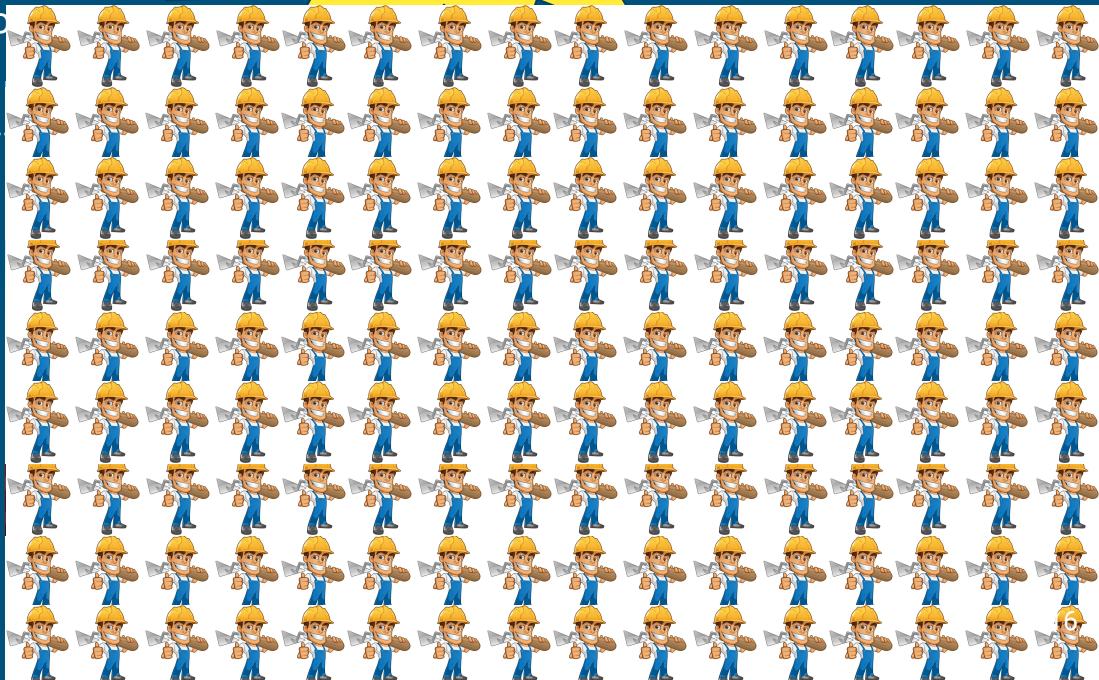


1. Introdução à programação paralela

- Programação paralela?

- Construir um muro muito longo
- 100km de comprimento!
- **100.000** pedreiros disponíveis.
- Cada pedreiro: **1**m de muro
- Terminará o muro: **??** dia

100m/dia



1. Introdução à programação paralela

- Programação paralela?

- Construir um muro muito longo
- 100km de comprimento!
- **100.000** pedreiros disponíveis.
- Cada pedreiro: **1**m de muro
- Terminará o muro: **??** dia
- **Material para todos trabalhar?**



100m/dia



1. Introdução à programação paralela

- Programação paralela?

- Muro = problema a ser resolvido
- Pedreiros = processadores (cores)
- Tijolos = dados
- Calçada = rede de comunicação entre os processadores
- Ajudante, servente, carrinho de mão = memória disponível (L1, L2, Ln, memória principal)
- Quanto mais pedreiros -> menos tempo gasto = escalabilidade
- Velocidade do pedreiro (100m/dia) = velocidade dos processadores
- Limite (físico) de pedreiros para cada tipo de problema = limite (físico) de processadores para cada tipo de problema (Granularidade)



1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?

1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



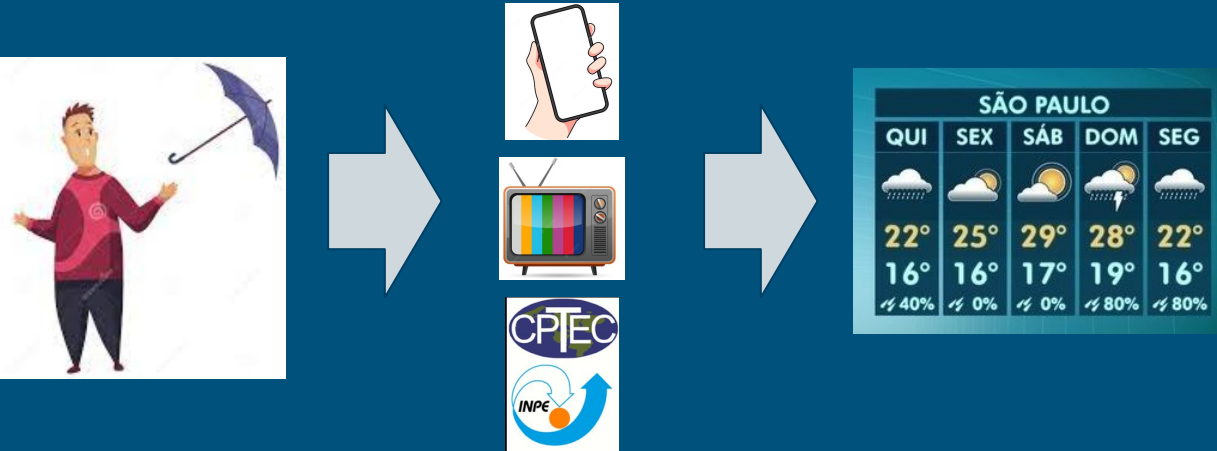
1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



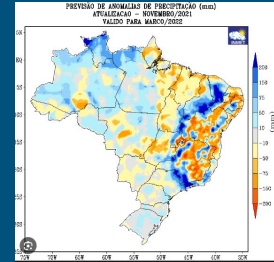
1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



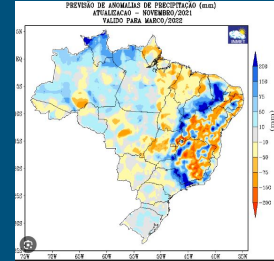
1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



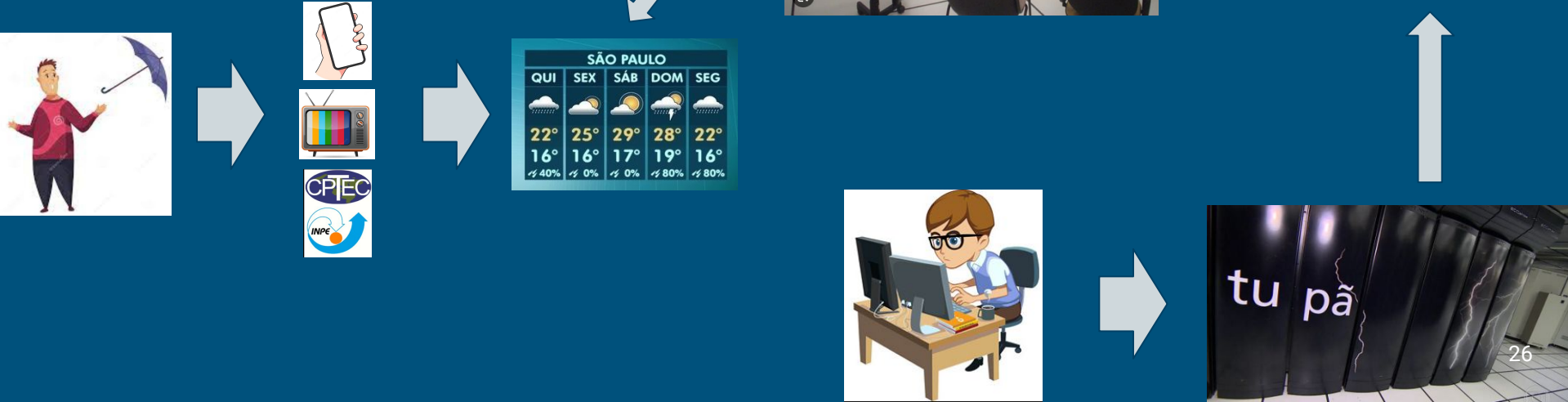
1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



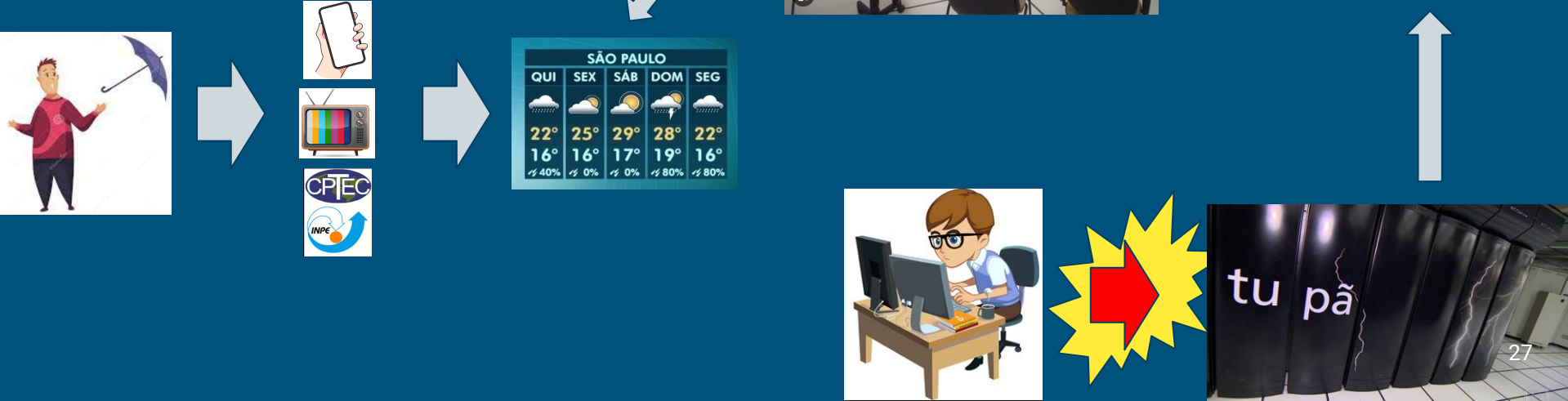
1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



Supercomputadores



Supercomputadores

| | Cray XC50 | 8 gabinetes do TUPA |
|--|---------------------------------------|---|
| Número de gabinetes | 1 | 8 |
| Número de Chassis por gabinete | 2 de 3 | 3 |
| Número de módulos por Chassi (max) | 16 | 8 |
| Número total de Módulos | 28 | 192 |
| Módulos computacionais | 26 | 182 |
| Módulos de serviço de entrada e saída | 2 | 10 |
| Nós computacionais configurados como serviços (Tier2, MOM) | 2 | 0 |
| Total de nós computacionais | 102 | 728 |
| Tipo de processador | Intel Skylake 6148 20 core 2.4 GHz | AMD Opteron 6172 2.1 Ghz "Magny-Cours" |
| Tipo de Memória | 16GB 2666Mhz DDR4 | 4GB 1600Mhz DDR3 |
| Memória por nó | 192 GB | 32GB |
| Número total de soquetes de processadores | 204 | 1.456 |
| Número de núcleos por soquete | 20 | 12 |
| Número total de núcleos | 4.080 | 17.472 |
| Velocidade do Processador | 2.4 GHz | 2.1 GHz |
| Performance total em Tflops | 313,3 | 146,8 |

Supercomputadores



XE6

1456 processadores
17.472 núcleos

146 Teraflops de
processamento (pico)

860 Terabytes de
armazenamento
para saída de dados

250~270KW de
Consumo de energia



XC50

208 processadores
4160 núcleos

313 Teraflops de
processamento (pico)

1 Petabyte de
armazenamento
para saída de dados

110KW de consumo
de energia

HPE CRAY SUPERCOMPUTERS WIN THE “EXASCALE TRIPLE-CROWN”



Argonne National Laboratory “Aurora”

- \$500M system subcontract
- HPE Cray EX 166 Rack Cabinets
- More than 2 EF sustained performance
- 10,124 Compute Node
- 21,248 Intel® Xeon® CPU Max Series
- 63,744 Intel Data Center GPU Max Series
- Slingshot switch
- 230 PB Usable Capacity, 25TB/s Cray ClusterStor E1000 storage
- Mixed AI and HPC workload



Oak Ridge National Laboratory “Frontier”

- \$600M system contract
- HPE Cray EX 74 Rack Cabinets
- More than 1.5 EF sustained performance
- 9,472 AMD Epyc 7453s "Trento" 64 core 2 GHz CPUs
- 37,888 AMD Instinct MI250X GPUs;
- Slingshot switch
- 700 PB Usable Capacity, 5PB/s Cray ClusterStor E1000 storage
- Mixed AI and HPC workload



Lawrence Livermore National Laboratory “El Capitan”

- \$600M system contract
- HPE Cray EX
- More than 1.5 EF sustained performance
- AMD MI300 APU; Slingshot switch
- 400 PB of Cray ClusterStor E1000 storage
- Mixed AI and HPC workload

Top 500:

top500.org/lists/top500/

@June/23

| Rank | System | Cores | Rmax (PFlop/s) | Rpeak (PFlop/s) | Power (kW) |
|------|---|-----------|-------------------|--------------------|---------------|
| 1 | Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States | 8,699,904 | 1,194.00 | 1,679.82 | 22,703 |
| 2 | Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan | 7,630,848 | 442.01 | 537.21 | 29,899 |
| 3 | LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland | 2,220,288 | 309.10 | 428.70 | 6,016 |
| 4 | Leonardo - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy | 1,824,768 | 238.70 | 304.47 | 7,404 |
| 5 | Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States | 2,414,592 | 148.60 | 200.79 | 10,096 |

1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?
 - Como é programada?
 - O que é um modelo de previsão?
 - ...

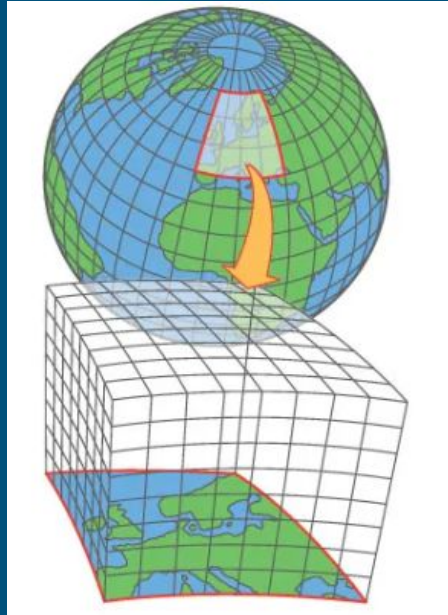
1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



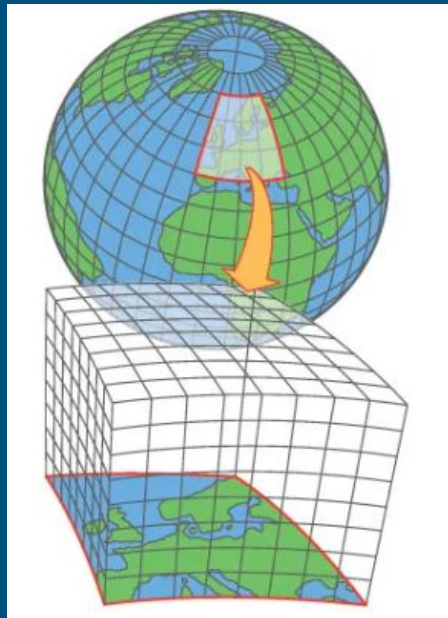
1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



$$\begin{aligned} \text{Continuidade: } \frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} &= 0 \\ \text{Momento em } x: \frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} + \frac{\partial(\rho uw)}{\partial z} &= -\frac{\partial P}{\partial x} + \frac{1}{Re} \left[\frac{\partial(\tau_{xx})}{\partial x} + \frac{\partial(\tau_{xy})}{\partial y} + \frac{\partial(\tau_{xz})}{\partial z} \right] \\ \text{Momento em } y: \frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho uv)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} + \frac{\partial(\rho vw)}{\partial z} &= -\frac{\partial P}{\partial y} + \frac{1}{Re} \left[\frac{\partial(\tau_{xy})}{\partial x} + \frac{\partial(\tau_{yy})}{\partial y} + \frac{\partial(\tau_{yz})}{\partial z} \right] \\ \text{Momento em } z: \frac{\partial(\rho w)}{\partial t} + \frac{\partial(\rho uw)}{\partial x} + \frac{\partial(\rho vw)}{\partial y} + \frac{\partial(\rho w^2)}{\partial z} &= -\frac{\partial P}{\partial z} + \frac{1}{Re} \left[\frac{\partial(\tau_{xz})}{\partial x} + \frac{\partial(\tau_{yz})}{\partial y} + \frac{\partial(\tau_{zz})}{\partial z} \right] \\ \text{Energia: } \frac{\partial(E_T)}{\partial t} + \frac{\partial(uE_T)}{\partial x} + \frac{\partial(vE_T)}{\partial y} + \frac{\partial(wE_T)}{\partial z} &= -\frac{\partial(uP)}{\partial x} - \frac{\partial(vP)}{\partial y} - \frac{\partial(wP)}{\partial z} - \frac{1}{RePr} \left[\frac{\partial(q_x)}{\partial x} + \frac{\partial(q_y)}{\partial y} + \frac{\partial(q_z)}{\partial z} \right] \end{aligned}$$

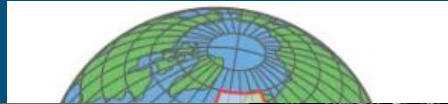
1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?

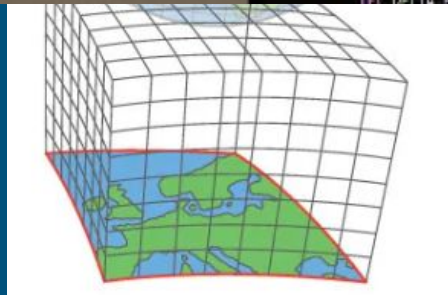


domínio do

Fortran



```
20 READ(*, *, ERR=20)A, B, C
C
DELTA= B**2-4.*A*C
C
IF( DELTA.GT.0 )THEN      ) (DUAS RAIZES REAIS)
  X1= ( -B-SQRT(DELTA) ) / ( 2.*A )
  X2= ( -B+SQRT(DELTA) ) / ( 2.*A )
  PRINT *, "RAIZES: X1= ", X1
  PRINT *, "X2= ", X2
ELSE
  IF( DELTA.EQ.0 ) THEN ) (DUAS RAIZES REAIS IGUAIS)
```

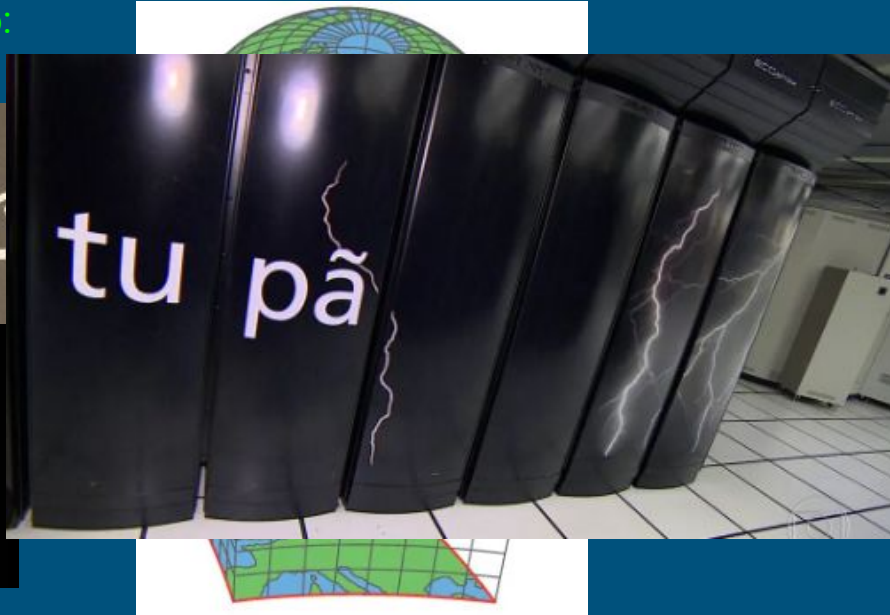


$$\begin{aligned} &= -\frac{\partial P}{\partial x} + \frac{1}{Re} \left[\frac{\partial(\tau_{xx})}{\partial x} + \frac{\partial(\tau_{xy})}{\partial y} + \frac{\partial(\tau_{xz})}{\partial z} \right] \\ &= -\frac{\partial P}{\partial y} + \frac{1}{Re} \left[\frac{\partial(\tau_{xy})}{\partial x} + \frac{\partial(\tau_{yy})}{\partial y} + \frac{\partial(\tau_{yz})}{\partial z} \right] \\ &= -\frac{\partial P}{\partial z} + \frac{1}{Re} \left[\frac{\partial(\tau_{xz})}{\partial x} + \frac{\partial(\tau_{yz})}{\partial y} + \frac{\partial(\tau_{zz})}{\partial z} \right] \end{aligned}$$

Energia: $\frac{\partial(E_T)}{\partial t} + \frac{\partial(uE_T)}{\partial x} + \frac{\partial(vE_T)}{\partial y} + \frac{\partial(wE_T)}{\partial z} = -\frac{\partial(uP)}{\partial x} - \frac{\partial(vP)}{\partial y} - \frac{\partial(wP)}{\partial z} - \frac{1}{RePr} \left[\frac{\partial(q_x)}{\partial x} + \frac{\partial(q_y)}{\partial y} + \frac{\partial(q_z)}{\partial z} \right]$

1. Introdução à programação paralela

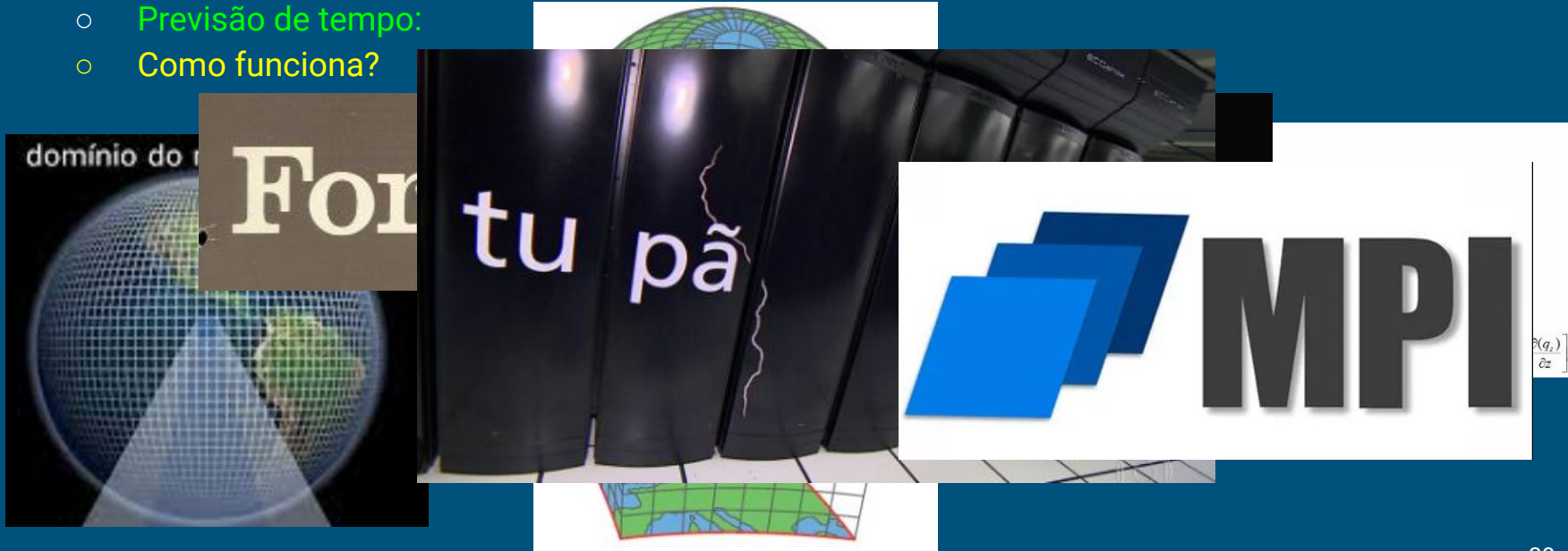
- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



$$\begin{aligned} &= -\frac{\partial P}{\partial x} + \frac{1}{Re} \left[\frac{\partial(\tau_{xy})}{\partial x} + \frac{\partial(\tau_{xy})}{\partial y} + \frac{\partial(\tau_{xz})}{\partial z} \right] \\ &= -\frac{\partial P}{\partial y} + \frac{1}{Re} \left[\frac{\partial(\tau_{xy})}{\partial x} + \frac{\partial(\tau_{yy})}{\partial y} + \frac{\partial(\tau_{yz})}{\partial z} \right] \\ &= -\frac{\partial P}{\partial z} + \frac{1}{Re} \left[\frac{\partial(\tau_{xz})}{\partial x} + \frac{\partial(\tau_{yz})}{\partial y} + \frac{\partial(\tau_{zz})}{\partial z} \right] \\ \tau_r &= -\frac{\partial(uP)}{\partial x} - \frac{\partial(vP)}{\partial y} - \frac{\partial(wP)}{\partial z} - \frac{1}{RePr} \left[\frac{\partial(q_x)}{\partial x} + \frac{\partial(q_y)}{\partial y} + \frac{\partial(q_z)}{\partial z} \right] \end{aligned}$$

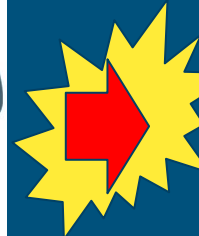
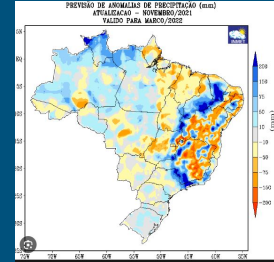
1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?

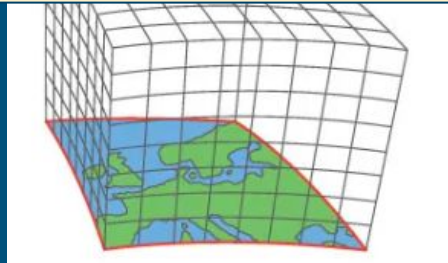


1. Introdução à programação paralela

- Exemplo real?
 - Previsão de tempo:
 - Como funciona?



Computação Científica



1. Introdução à programação paralela - MPI



MPI - *Message Passing Interface*

- MPI é um padrão de protocolo de troca de mensagens entre **processadores**, (biblioteca)
 - Processadores **x** cores (ranks) **x** processos ??
- Funciona em ambientes de memória distribuída
 - Cada core executa um processo como se fosse uma cópia do programa separadamente um do outro, independente.
 - Memórias distintas

1. Introdução à programação paralela - MPI



MPI - *Message Passing Interface*

- mpi-forum.org
- MPI-2 : 1997
- MPI-2.1 : 2008
- MPI-2.2 : 2009
- MPI-3 : 2012
- MPI-3.1 : 2015/16
- Atualmente versão 4.1 (5.0 soon)
- Muitas implementações:
 - mpich - MPI CHameleon (OpenSource) == mpich.org
 - openMPI (OpenSource)
 - IMPI - Intel MPI (Private)
 - Cray MPI (Private)
 - Bullx MPI
 - ...

1. Introdução à programação paralela - MPI

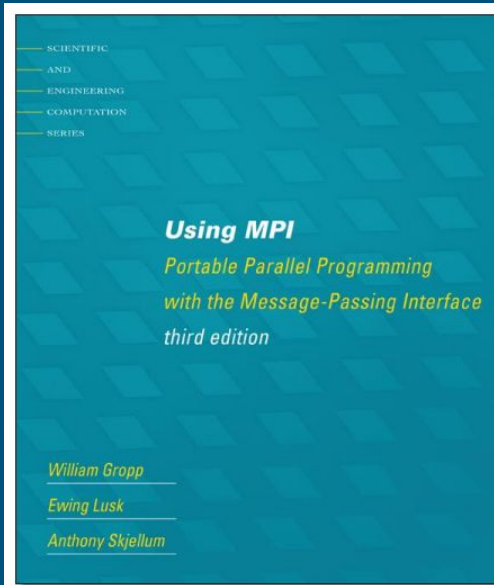
MPI - *Message Passing Interface*



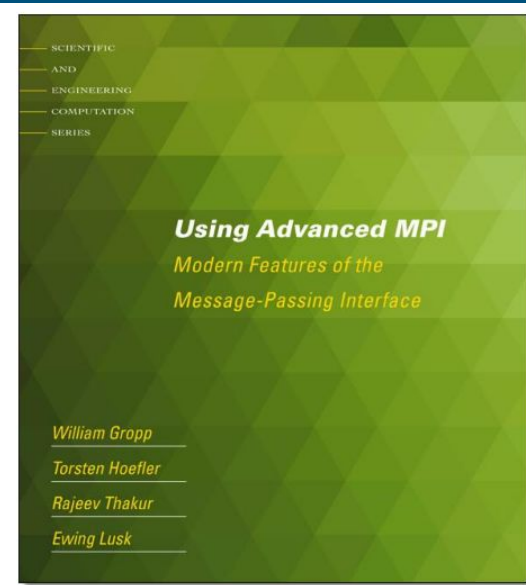
- mpi-forum.org
- MPI-2 : 1997
- MPI-2.1 : 2008
- MPI-2.2 : 2009
- MPI-3 : 2012
- MPI-3.1 : 2015/16
- Atualmente versão 4.1 (5.0 soon)
- Muitas implementações:
 - **mpich - MPI CHameleon (OpenSource) == mpich.org**
 - openMPI (OpenSource)
 - IMPI - Intel MPI (Private)
 - Cray MPI (Private)
 - Bullx MPI
 - ...

1. Introdução à programação paralela - MPI

Bibliografia:



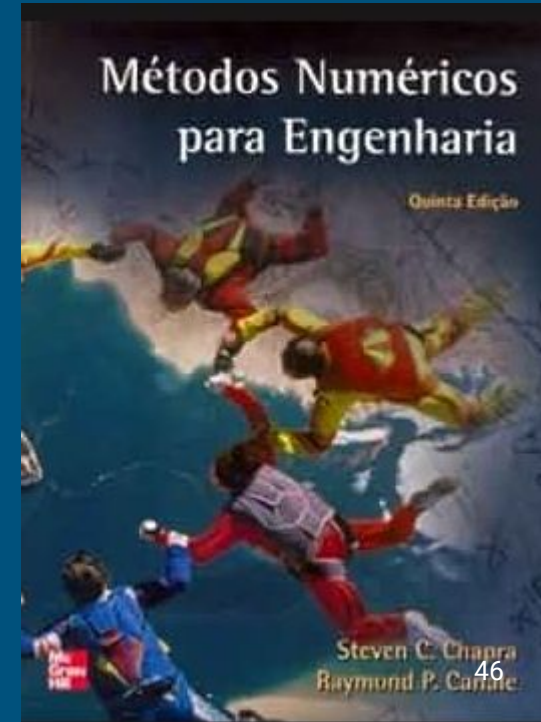
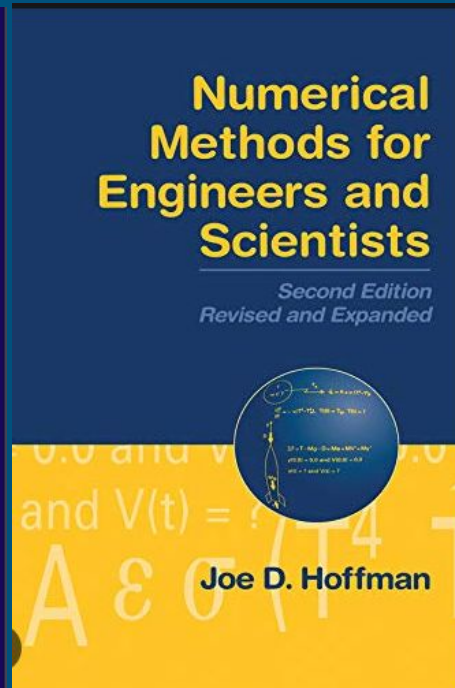
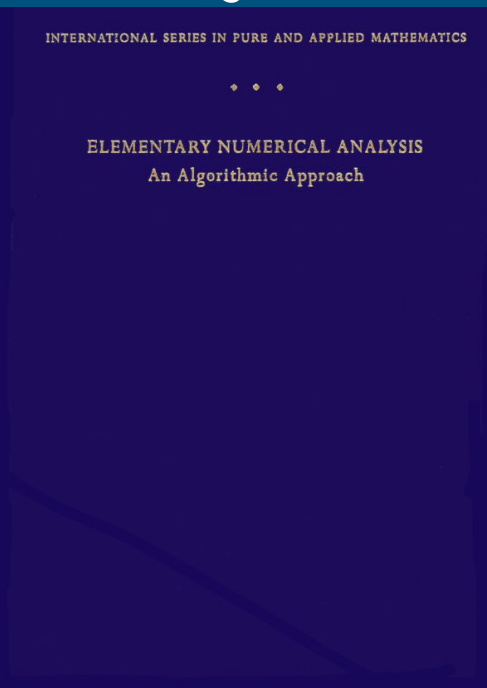
Basic MPI



Advanced MPI, including MPI-3

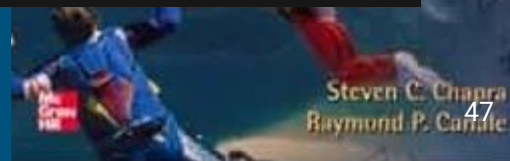
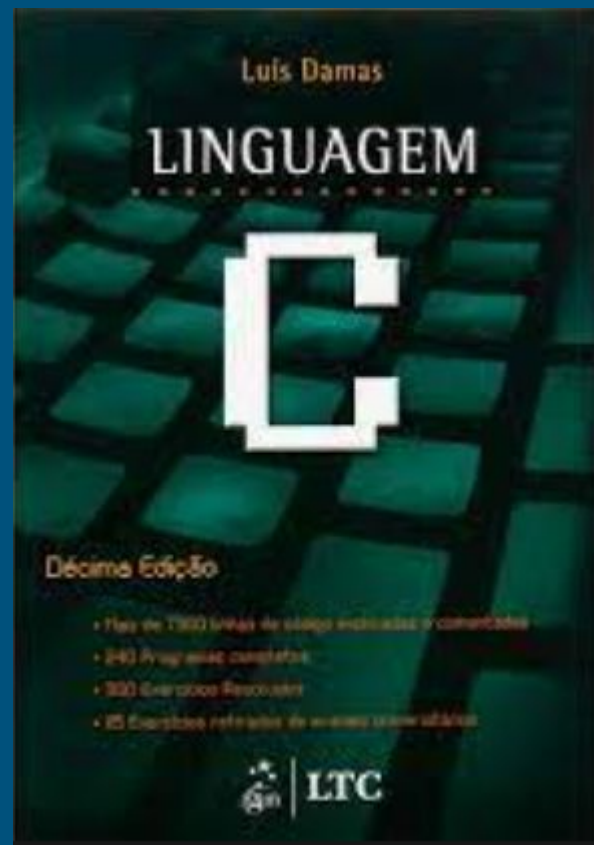
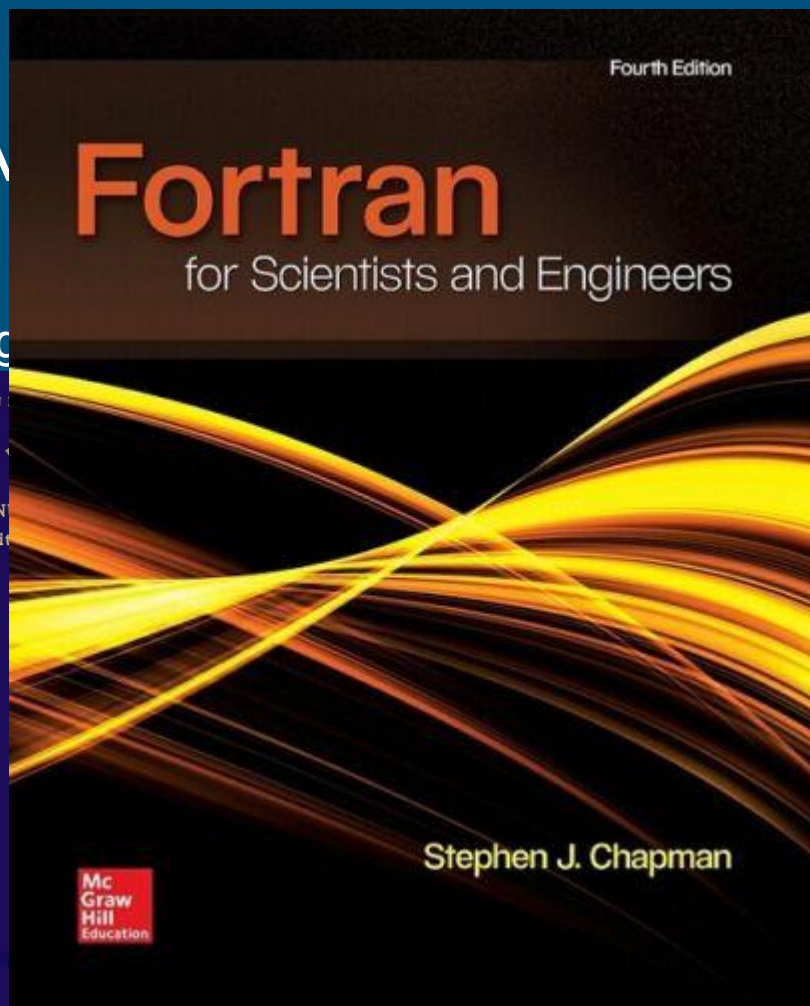
1. Métodos numéricos

Bibliografia:



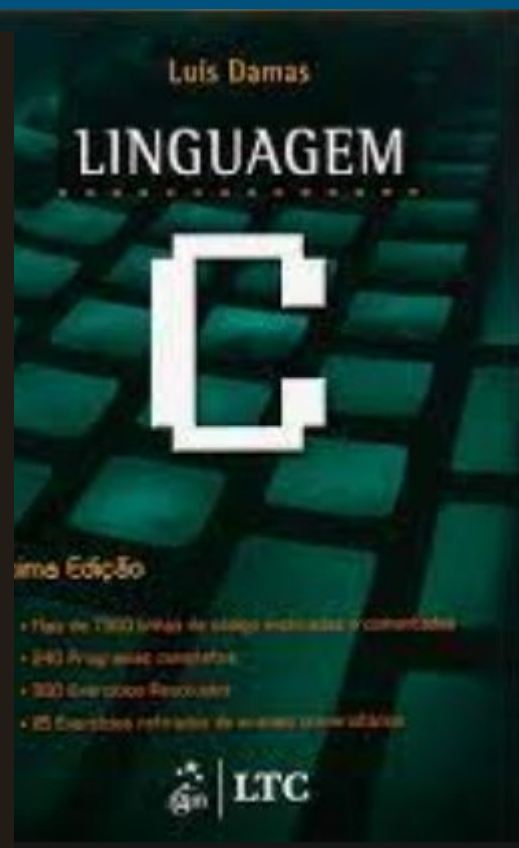
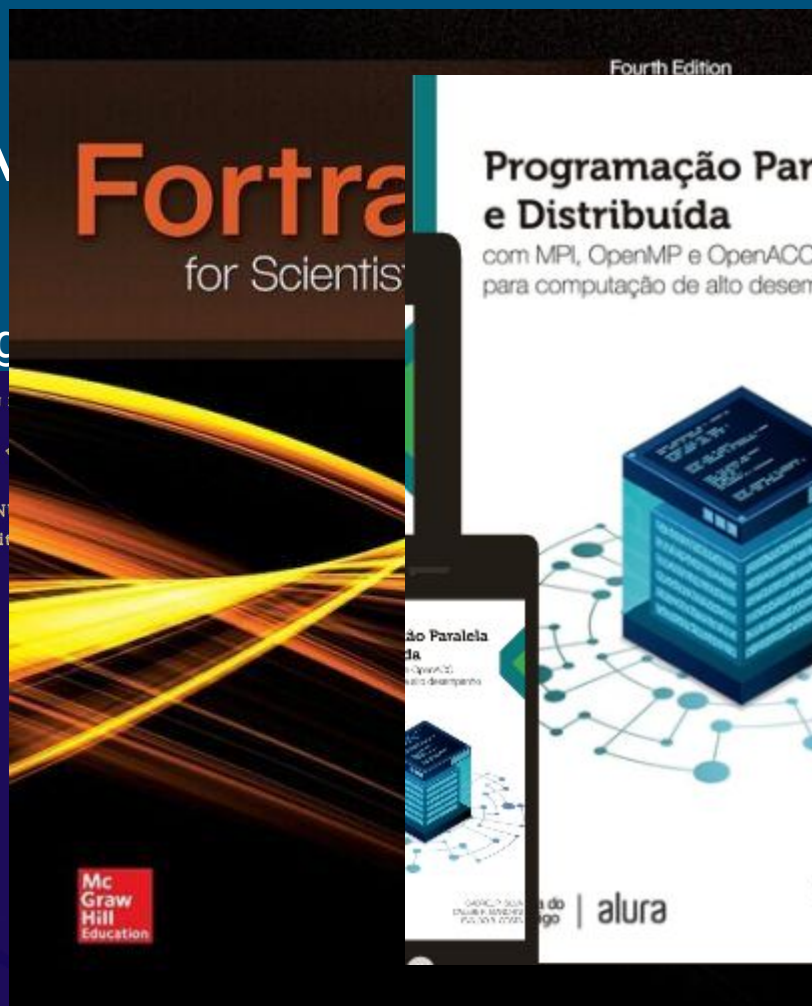
1. M

Bibliog



1. M

Bibliog



Steven C. Chao
Raymond P. Canale

1. Introdução à programação paralela - MPI

MPI - *Message Passing Interface*



- MPICH - MPI CHameleon == mpich.org
 - mpich.org/static/docs == manual online atualizado
 - MPI oferece cerca de 400 rotinas e funções!!!
 - Com +- 15 já é suficiente para paralelizar a maioria dos problemas!

Para instalar no seu computador (Linux@Ubuntu):

```
$ sudo apt install mpich
```

1. Introdução à programação paralela - MPI

MPI - *Message Passing Interface*

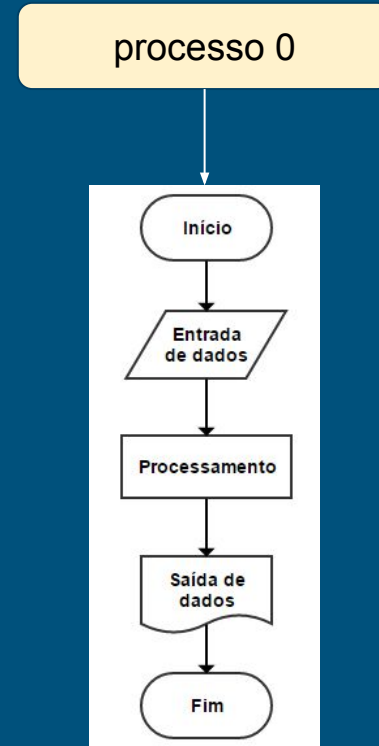


- MPICH

- Compilar seu código **C**: `mpicc seuCodigo.c -o seuCodigo.x`
- Executar seu código **C**: `mpirun/mpiexec -n/-np NC seuCodigo.x`
 - **NC** :: número de processos MPI desejado
- Compilar seu código **Fortran**: `mpif90 seuCodigo.f90 -o seuCodigo.x`
- Executar seu código **Fortran**: `mpirun/mpiexec -n/-np NC seuCodigo.x`
 - **NC** :: número de processos MPI desejado

1. Introdução à programação paralela - MPI

Estrutura geral de um programa **sequencial**:



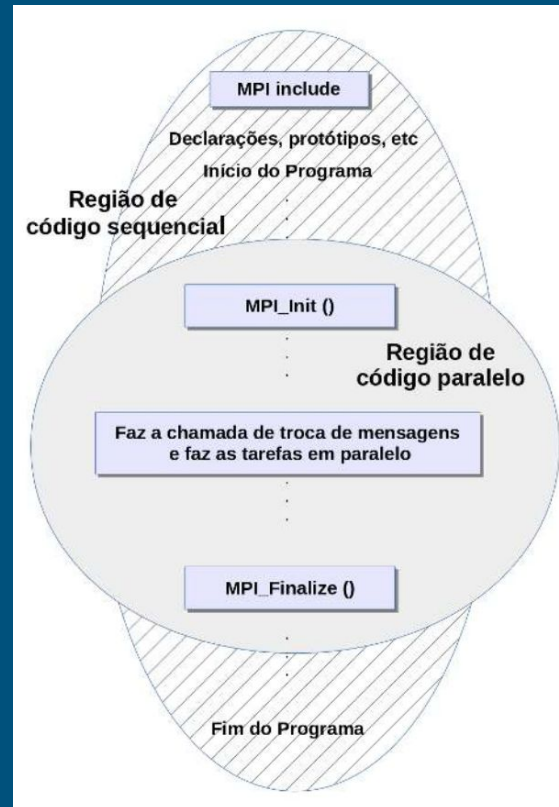
1. Introdução à programação paralela - MPI

Estrutura geral de um programa **paralelo MPI**:

`MPI_Init();`

região paralela, todos os cores vão executar essa região

`MPI_Finalize();`



1. Introdução à programação paralela - MPI

Estrutura geral de um programa paralelo MPI:

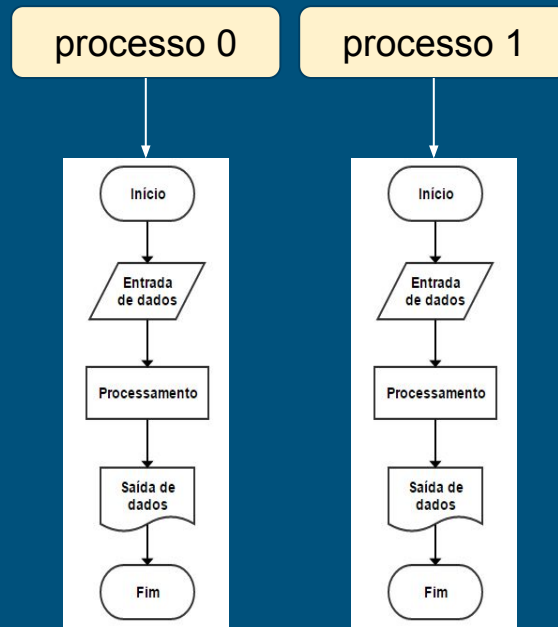
```
$ mpirun -n 2 meuCodigo.x
```

Cada processo recebe uma **cópia** do programa.

Cada processo **executa** uma **cópia** do programa!

Respeitando a região sequencial/paralela!

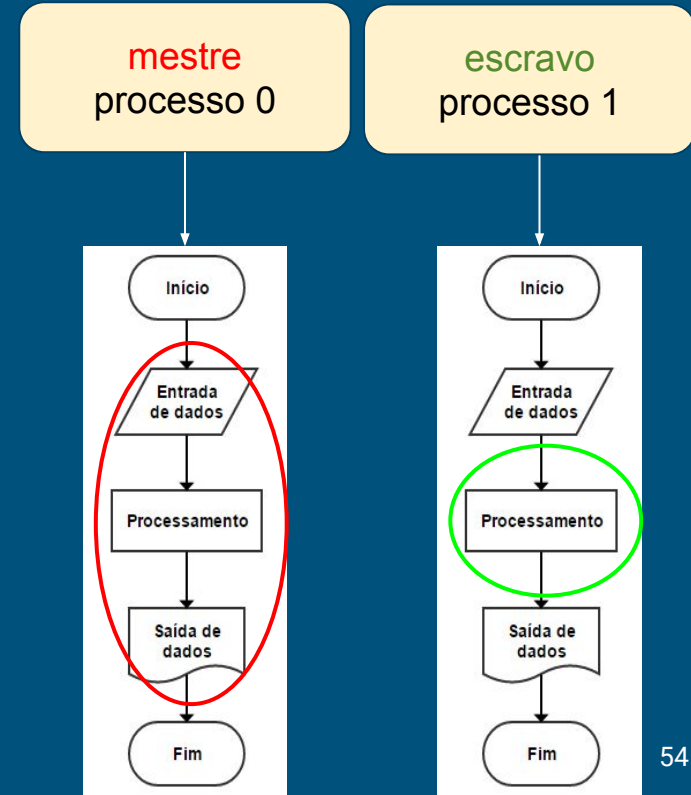
Um processo não acessa a memória do outro.



1. Introdução à programação paralela - MPI

Mestre-escravo:

- P0 lê dados de entrada e **distribui** para os outros;
- P1, P2, ..., Pn processa o seu domínio de dados e **devolve** para P0;

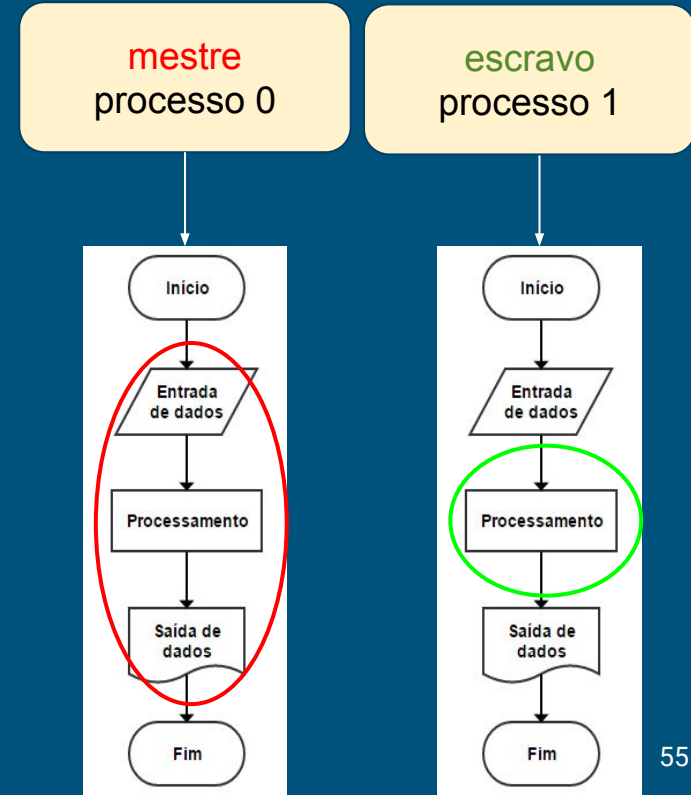


1. Introdução à programação paralela - MPI

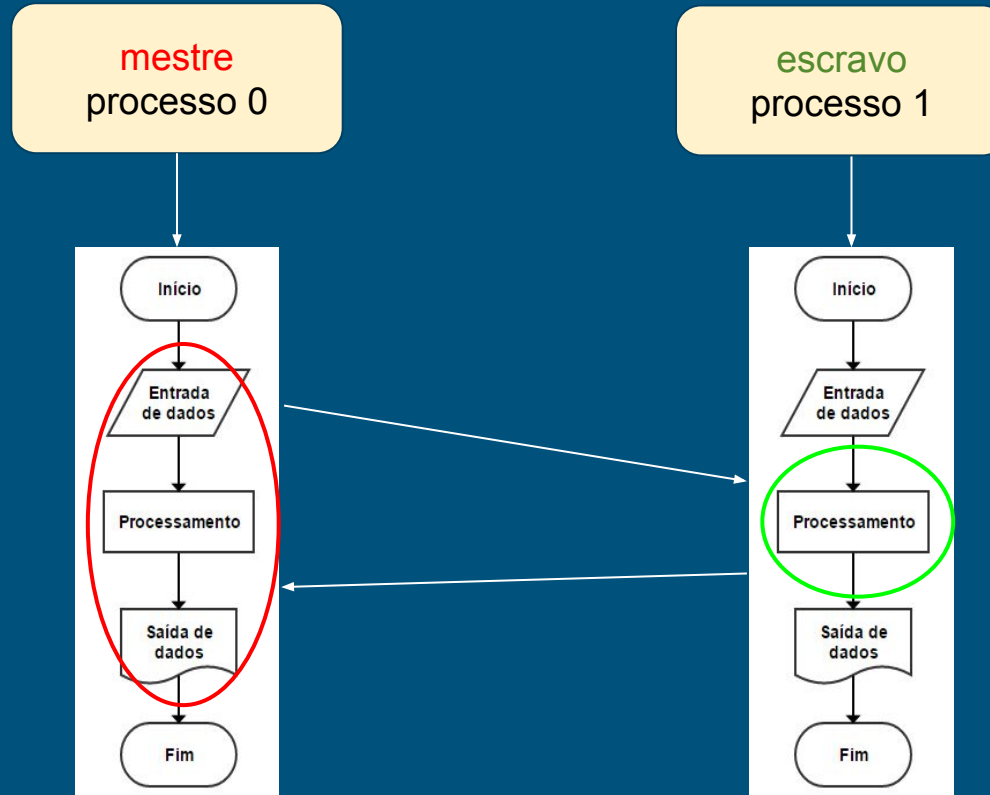
Mestre-escravo:

- P0 lê dados de entrada e **distribui** para os outros;
- P1, P2, ..., Pn processa o seu domínio de dados e **devolve** para P0;

Comunicação entre processos (cores)(ranks)!!!



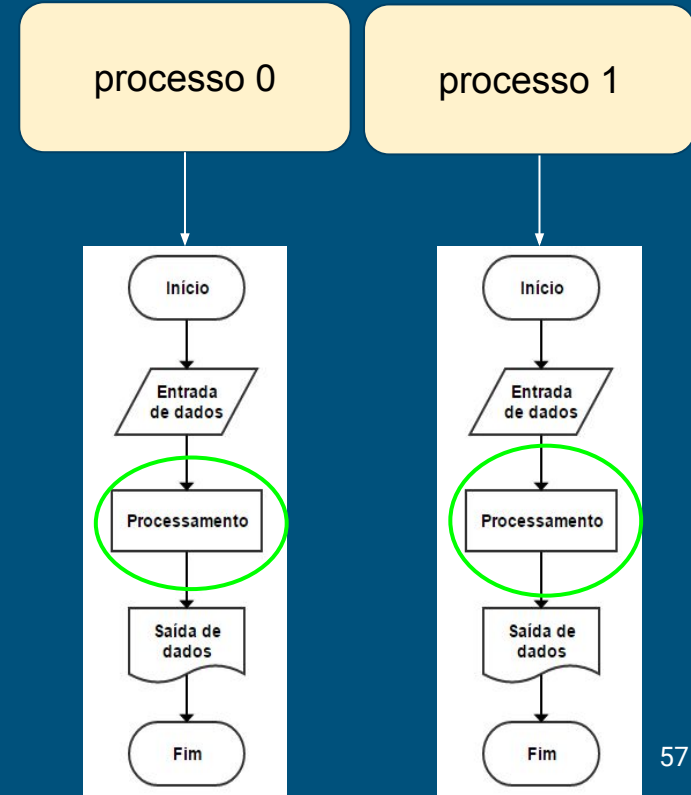
1. Introdução à programação paralela - MPI



1. Introdução à programação paralela - MPI

Independente:

- Cada processo lê seu próprio domínio de dados;
- Todos os procs recebem a mesma qtde de dados;
- Cada processo processa seu pedaço do problema e somente **envia** para P0;
- P0 **reúne** todos os dados e faz IO;
- Cada processo escreve sua saída em PIO (Parallel IO)
 - Bibliotecas de PIO
 - MPI-PIO



1. Introdução à programação paralela - MPI

Comunicação MPI:

1. Two-sided comm;
 - a. 99% dos casos de programação paralela com MPI

2. One-sided comm; (+- Nova, padrão 3.1/ -> 2015)
 - a. 1%, nível hard

1. Introdução à programação paralela - MPI

Comunicação MPI:

1. Two-sided comm;

- Ponto-a-ponto:
 - Conversa entre 2 pessoas no zap, telefone, etc...
 - 1 fala, outro ouve; Outro fala, o primeiro ouve;
- Coletiva:
 - Conversa num grupo no zap, professor em sala de aula, etc...
 - Um fala para um grupo de pessoas, e o grupo todo ouve a mensagem;

1. Introdução à programação paralela - MPI

Comunicação MPI:

2. One-sided comm;

- RMA Put/get (padrão 2.5/)
 - Memória compartilhada, todos processos acessam a mesma região de memória;
 - Um escreve a mensagem numa região da memória, e fica disponível para todos os outros processos;
 - Cada processo pode acessar um dado na memória escrito por outro processo sem que ele perceba!
 - O acesso ao dado ainda passa pela biblioteca MPI, usa o buffer, ainda há tráfego de dado!
- MPI-SHM (padrão 3.1/)
 - Idem ao Put/get, porém o dado não passa pela biblioteca MPI, não há tráfego de dados, o acesso é diretamente à memória via endereço de memória (ponteiros)!

1. Introdução à programação paralela - MPI

1. Two-sided comm;

- Ponto-a-ponto:
 - Envio: `MPI_Send()`;
 - Recebimento: `MPI_Recv()`;

Comunicação:

- Blocante;
- Não-blocante;
- Síncrona;
- Assíncrona;

1. Introdução à programação paralela - MPI

Blocante:

- O processo que está recebendo a mensagem fica parado enquanto a mensagem está sendo enviada, e só é liberado para executar as próximas instruções depois que finalizar o recebimento.
- `MPI_Send()`; `MPI_Recv()`;

Ex: Uma ligação de telefone, onde eu paro para ouvir, e não faço nada até que a ligação termine.

1. Introdução à programação paralela - MPI

Não-blocante:

- O processo que recebe a mensagem não pára durante o envio, continua suas tarefas enquanto o envio está sendo feito.
- Nesse caso é necessário usar `MPI_Wait()` para confirmar se envio já terminou para então poder usar o dado enviado;
- `MPI_Isend()`; `MPI_Irecv()`;

Ex: Mensagem de zap, email; Eu sei que vou receber uma mensagem, mas eu continuo fazendo outras atividades e quando eu for precisar do dado enviado, eu páro e confirmo se o dado chegou completamente.

1. Introdução à programação paralela - MPI

Funções MPI:

1. `MPI_Init();` == inicializa
2. `MPI_Send();` == envia msg
3. `MPI_Recv();` == receb. msg
4. `MPI_Isend();` == envia Nblck
5. `MPI_Irecv();` == rec. Nblck
6. `MPI_Wait();` == espera com.
7. `MPI_Finalize();` == finaliza

1. Introdução à programação paralela - MPI

Funções MPI:

1. `MPI_Init();` == inicializa
2. `MPI_Send();` == envia msg
3. `MPI_Recv();` == receb. msg
4. `MPI_Isend();` == envia Nblck
5. `MPI_Irecv();` == rec. Nblck
6. `MPI_Wait();` == espera com.
7. `MPI_Finalize();` == finaliza

8. `MPI_Comm_rank();`

- Consultar o número do processo;

9. `MPI_Comm_size();`

- Consultar qtde total de processos utilizados.

1. Introdução à programação paralela - MPI

Funções MPI:

1. MPI_Init();
2. MPI_Send();
3. MPI_Recv();
4. MPI_Isend();
5. MPI_Irecv();
6. MPI_Wait();
7. MPI_Finalize();
8. MPI_Comm_rank();
9. MPI_Comm_size();

10. MPI_Bcast();

- Envia mensagem para todos os processos! Coletiva!

11. MPI_Wtime();

- Registra o horário de máquina;
- Muito útil para contabilizar tempo gasto em trechos do código;

1. Ambiente de desenvolvimento paralelo

Preparando seu ambiente (linux@ubuntu):

Instalando **MPIch**:

```
$ sudo apt install mpich build-essential
```

Instalando compilador **C**:

```
$ sudo apt install gcc
```

Para este curso:

Instalar o **git**:

```
$ sudo apt install git
```

1. Meu primeiro programa MPI

Objetivo:

- Hello world!
 - Cada processo vai imprimir na tela o seu “hello world”!
-
- Versão sequencial;
 - Versão paralela;

1. Ex. 1 - seq:

Criar o seu próprio!

Compilar:

```
$ gcc helloWorld.c -o helloWorld.x
```

Executar:

```
$ ./helloWorld.x
```

```
1  #include<stdio.h>
2
3  int main(int argc, char const *argv[])
4  {
5      printf("\n Hello World \n");
6
7
8      return 0;
9  }
10
```

1. Ex. 1 - seq:

Baixar do github:

```
$ git clone https://github.com/carlosrenatosouza2/Mini\_Curso\_MPI.git
```

```
$ cd Mini_Curso_MPI/Ex1_helloWorld/sequencial/
```

```
$ ls -ltr
```

comp_hello_word_seq.ksh

hello_world_seq.c

Compilar:

```
$ ./comp_hello_word_seq.ksh
```

Executar:

```
$ ./hello_world_seq.x
```

```
1  #include<stdio.h>
2
3  int main(int argc, char const *argv[])
4  {
5      printf("\n Hello World \n");
6
7
8      return 0;
9  }
10
```

1. Ex. 1 - mpi:

Baixar do github:

```
$ cd Mini_Curso_MPI/Ex1_helloWorld/mpi/v0
```

```
$ ls -ltr
```

```
comp_hello_word_MPI.ksh
```

```
hello_world_MPI.c
```

```
run_hello_word_MPI.ksh
```

```
1  #include <stdio.h>
2  #include <mpi.h>
3
4  int main(int argc, char const *argv[])
5  {
6
7      MPI_Init(NULL, NULL);
8
9
10     printf("\n Hello World \n");
11
12     MPI_Finalize();
13
14     return 0;
15 }
16
```

1. Ex. 1 - mpi:

Baixar do github:

```
$ cd Mini_Curso_MPI/Ex1_helloWorld/mpi/v0
```

```
$ ls -ltr
```

```
comp_hello_word_MPI.ksh
```

```
hello_world_MPI.c
```

```
run_hello_word_MPI.ksh
```

```
1  #include <stdio.h>
2  #include <mpi.h>
3
4  int main(int argc, char const *argv[])
5  {
6
7      MPI_Init(NULL, NULL);
8
9
10     printf("\n Hello World \n");
11
12     MPI_Finalize();
13
14     return 0;
15 }
16
```

1. Ex. 1 - mpi:

Baixar do github:

```
$ cd Mini_Curso_MPI/Ex1_helloWorld/mpi/v0
```

```
$ ls -ltr
```

```
comp_hello_word_MPI.ksh
```

```
hello_world_MPI.c
```

```
run_hello_word_MPI.ksh
```

```
1  #include <stdio.h>
2  #include <mpi.h>
3
4  int main(int argc, char const *argv[])
5  {
6      MPI_Init(NULL, NULL);
7
8      printf("\n Hello World \n");
9
10     MPI_Finalize();
11
12     return 0;
13 }
14
15
16
```

1. Ex. 1 - mpi:

Baixar do github:

```
$ cd Mini_Curso_MPI/Ex1_helloWorld/mpi/v0
```

```
$ ls -ltr
```

```
comp_hello_word_MPI.ksh
```

```
hello_world_MPI.c
```

```
run_hello_word_MPI.ksh
```

```
1 #include <stdio.h>
2 #include <mpi.h>
3
4 int main(int argc, char const *argv[])
5 {
6
7     MPI_Init(NULL, NULL);
8
9
10    printf("\n Hello World \n");
11
12    MPI_Finalize();
13
14    return 0;
```

```
1 #!/bin/ksh
2
3 codname="hello_world_MPI"
4
5 rm -f ${codname}.x
6
7 echo "Compilando..."
8 mpicc ${codname}.c -o ${codname}.x
9 echo "Compilado!  ${codname}.x"
10
11
12
13 if [ ! -s ${codname}.x ]
14 then
15     echo "Deu ruim..."
16 fi
```

1. Ex. 1 - mpi

Baixar do github:

\$ cd Mini_Curso_MPI/Ex1_MPI

\$ ls -ltr

comp_hello_word_MPI.ksh

hello_world_MPI.c

run_hello_word_MPI.ksh

```
1  #!/bin/ksh
2
3  if [ $# -ne 1 ]
4  then
5      echo ""
6      echo "usage: ${0} [NP]"
7      echo ""
8      echo "where:"
9      echo ""
10     echo "NP: total_ranks_MPI :: [ ] total number of MPI ranks/processes"
11     echo ""
12     exit
13 fi
14
15 np=${1}
16 codname="hello_world_MPI"
17
18
19 if [ ! -s ${codname}.x ]
20 then
21     echo "Executável nao existe. ${codname}.x"
22     exit
23 fi
24
25 mpirun -n ${np} ${codname}.x
26
27 fi
```

1. Ex. 1 - mpi:

Sem script:

```
$ cd Mini_Curso_MPI/Ex1_helloWorld/mpi/v0
```

Compilar:

```
$ mpicc hello_world_MPI.c -o hello_world_MPI.x
```

Executar:

```
$ mpirun -n 2 hello_world_MPI.x
```



1. Ex. 1 - mpi:

Sem script:

```
$ cd Mini_Curso_MPI/Ex1_helloWorld/mpi/v0
```

Compilar:

```
$ mpicc hello_world_MPI.c -o hello_world_MPI.x
```

Executar:

```
$ mpirun -n 2 hello_world_MPI.x
```



**Varie a qtde de
processos!**

Qual foi o resultado?

1. Ex. 1 - mpi

Sem script:

\$ cd Mini_Curso_MPI/Ex1_helloW

Compilar:

\$ mpicc hello_world_MPI2.c -o he

Executar:

\$ mpirun -n 2 hello_world_MPI2.x

C hello_world_MPI2.c M x

Ex1_helloWorld > mpi > v1 > C hello_world_MPI2.c > main(int, char const * [])

```
1  #include <stdio.h>
2  #include <mpi.h>
3
4  int main(int argc, char const *argv[])
5  {
6      int r, p;
7
8      // Inicializa o MPI: inicio da regio paralela.
9      MPI_Init(NULL, NULL);
10
11     // Qual processo sou eu?
12     // Consulta o numero do meu processo:
13     MPI_Comm_rank(MPI_COMM_WORLD, &r);
14
15     // Consulta quantos processos existem no total:
16     MPI_Comm_size(MPI_COMM_WORLD, &p);
17
18     printf("\n Hello World from rank %d of %d processes.\n", r, p);
19
20     // Finaliza o MPI: fecha a regio paralela.
21     MPI_Finalize();
22
23     return 0;
24 }
```

Comunicação ponto-a-ponto

Comunicação básica bloqueante:

- Enquanto um processo realiza uma operação de envio, o outro processo realiza uma operação de recebimento da mensagem;
- Funções: `MPI_Send()`; e `MPI_Recv()`;
- Comunicação de forma Bloqueante;
- Ou seja, o programa não segue em frente enquanto a operação de receber/enviar estiver completa!

Comunicação ponto-a-ponto

C binding

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,  
             int tag, MPI_Comm comm)
```

Argumentos:

1. Endereço do dado a ser transmitido;
2. Número de itens a ser enviado;
3. Tipo do dado;
4. Destino;
5. Tag, para identificar a mensagem;
6. Comunicador;

Comunicação ponto-a-ponto

C binding

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,  
             int tag, MPI_Comm comm)
```

Argumentos:

1. Endereço do dado a ser transmitido;
2. Número de itens a ser enviado;
3. Tipo do dado;
4. Destino;
5. Tag, para identificar a mensagem;
6. Comunicador;



Comunicador MPI:

Um grupo de processos (ranks/cores) que se comunicam entre si.

MPI_COMM_WORLD :: comunicador global, abrange todos os processos disponíveis.

É possível criar outros comunicadores, dividindo os processos em grupos correlatos, e ainda criar sub comunicadores.

Comunicação ponto-a-ponto

C binding

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,  
             int tag, MPI_Comm comm)
```

Exemplo:

```
int r, p;  
int dest=1, tag=1;  
int outmsg=1;  
  
MPI_Send(&outmsg, 1, MPI_INT, dest, tag, MPI_COMM_WORLD)
```

Comunicação ponto-a-ponto

C binding

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,  
             int tag, MPI_Comm comm, MPI_Status *status)
```

Argumentos:

1. Endereço do dado a ser recebido;
2. Número de itens a ser recebido;
3. Tipo do dado;
4. Fonte;
5. Tag, identificar a mensagem;
6. Comunicador;
7. Status da mensagem;

Comunicação ponto-a-ponto

C binding

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,  
             int tag, MPI_Comm comm, MPI_Status *status)
```

```
int r, p;  
int source=0, tag=1;  
int inmsg;  
MPI_Status stat;  
  
MPI_Recv(&inmsg, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &stat)
```


Comunicação ponto-a-ponto

Como que funciona na prática?

P0 envia para um dado para P1

P1 recebe o dado de P0

e

P1 envia outro dado pra P0

P0 recebe o dado de P1

Comunicação ponto-a-ponto

C binding

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,  
             int tag, MPI_Comm comm)
```

P0 envia para um dado para P1 → MPI_Send(&dado, 1, MPI_INT, 1, tag, MPI_COMM_WORLD)

P1 recebe o dado de P0 → MPI_Recv(&dado, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, st)

e

P1 envia outro dado pra P0 → MPI_Send(&dado, 1, MPI_INT, 1, tag, MPI_COMM_WORLD)

P0 recebe o dado de P1 → MPI_Recv(&dado, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, st)

Ping-Pong (bloqueio)

\$ cd Mini_Curso_MPI/Ex2_Pingpong/mpi/v0

Compilar:

\$ mpicc Ex2_mpi.c -o Ex2_mpi.x

Executar:

\$ mpirun -n 2 Ex2_mpi.x

```
int main(int argc, char const *argv[])
{
    int r, p;
    int source, dest, tag=1;
    int dado=0, ibuff;
    MPI_Status stat;

    MPI_Init(NULL, NULL);

    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    printf("\nAntes da com.: rank: %d, dado: %d.\n", r, dado);

    if ( r == 0 ){
        dest = 1;
        source = 1;
        ibuff=10;
        MPI_Recv(&dado, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &stat);
        MPI_Send(&ibuff, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
    }
    else {
        dest = 0;
        source = 0;
        ibuff=50;
        MPI_Send(&ibuff, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
        MPI_Recv(&dado, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &stat);
    }

    printf("\nDepois da com.: rank: %d, dado: %d.\n", r, dado);

    MPI_Finalize();

    return 0;
}
```

Ping-Pong (bloqueio)

\$ cd Mini_Curso_MPI/Ex2_Pingpong/mpi/v0

Compilar:

\$ mpicc Ex2_mpi.c -o Ex2_mpi.x

Executar:

\$ mpirun -n 2 Ex2_mpi.x

```
int main(int argc, char const *argv[])
{
    int r, p;
    int source, dest, tag=1;
    int dado=0, ibuff;
    MPI_Status stat;

    MPI_Init(NULL, NULL);

    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    printf("\nAntes da com.: rank: %d, dado: %d.\n", r, dado);

    if ( r == 0 ){
        dest = 1;
        source = 1;
        ibuff=10;
        MPI_Recv(&dado, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &stat);
        MPI_Send(&ibuff, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
    }
    else {
        dest = 0;
        source = 0;
        ibuff=50;
        MPI_Send(&ibuff, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
        MPI_Recv(&dado, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &stat);
    }

    printf("\nDepois da com.: rank: %d, dado: %d.\n", r, dado);

    MPI_Finalize();

    return 0;
}
```

Exemplo **sem bloqueio**

- O programa não fica esperando a mensagem chegar ou ser enviada
- Existe um mecanismo do MPI para controlar mensagens que ainda não chegaram/enviadas
- MPI_Wait();
- Comunicações não-bloqueantes são importantes para **sobrepôr comunicação com computação** e explorar possíveis ganhos de desempenho

Importante!

O programa somente deve acessar/utilizar/alterar o dado transmitido DEPOIS que a comunicação estiver completa!

Exemplo sem bloqueio

Mini_Curso_MPI/Ex3_CommNonBlk/mpi/v0

- MPI_IRecv sempre antes de MPI_Isend
- 99% dos programas MPI são implementados nesse modelo: Comunicação x computação
- MPI_Wait() :: muitas versões, muito flexível, existe 1 MPI_Wait para seu caso específico.

```
int main(int argc, char const *argv[])
{
    int r, p, esq, dir, buf[2], tag1=1, tag2=2;

    MPI_Status stats[4];
    MPI_Request reqs[4]; // controle para comunicacoes nao-bloqueantes

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    // Determinando os ranks vizinhos: a esquerda e a direita:
    esq = r - 1;
    dir = r + 1;
    if ( r == 0 )          esq = p - 1; // se sou rank 0, meu viz. a esq = ultimo
    if ( r == (p - 1) ) dir = 0;      // se sou o ultimo, meu viz. a dir = primeiro

    // Recebendo (postando) mensagens nao-bloqueantes para meus vizinhos:
    MPI_Irecv(&buf[0], 1, MPI_INT, esq, tag1, MPI_COMM_WORLD, &reqs[0]);
    MPI_Irecv(&buf[1], 1, MPI_INT, dir, tag2, MPI_COMM_WORLD, &reqs[1]);

    // Enviando (postando) mensagens nao-bloqueantes para meus vizinhos:
    MPI_Isend(&r, 1, MPI_INT, esq, tag2, MPI_COMM_WORLD, &reqs[2]);
    MPI_Isend(&r, 1, MPI_INT, dir, tag1, MPI_COMM_WORLD, &reqs[3]);

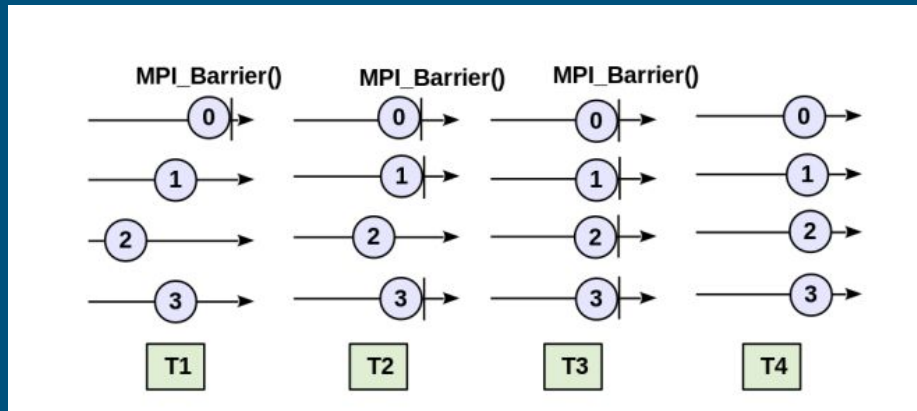
    //
    // Faco algum trabalho de computacao enquanto as mensagens sao transmitidas
    //

    // aguardando, verificando as mensagens chegarem:
    MPI_Waitall(4, reqs, stats);
    printf("\n Rank %d :: viz esq: %d viz dir: %d\n", r, buf[0], buf[1]);

    MPI_Finalize();
    return 0;
}
```

Comunicação Coletiva: Barrier

- `MPI_Barrier(Comunicador);`
 - fornece um mecanismo para sincronizar os processos pertencentes ao comunicador informado;
 - Cada processo fica parado (bloqueado) quando executa essa função;
 - Os processos são liberados (desbloqueados) somente quando todos chegarem no ponto da função;

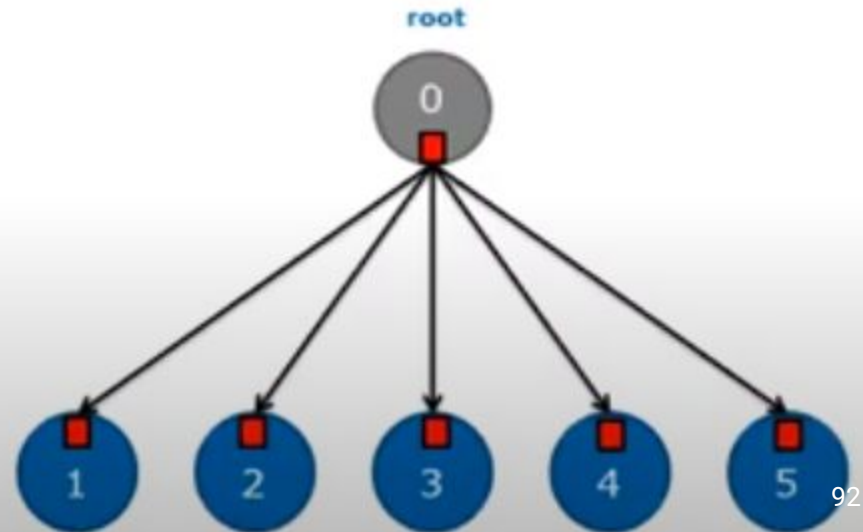


Comunicação Coletiva: **BroadCast**

C binding

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root,  
             MPI_Comm comm)
```

- Um rank envia para todos os outros ranks que fazem parte do comunicador
- Funciona como uma barreira também!
 - Todos os ranks ficam parados nesse ponto até que todos os ranks recebam a mensagem.



Exemplo BCast

\$ cd Mini_Curso_MPI/Ex4_ComColetiva/mpi/Bcast

Compilar:

\$ mpicc Ex4_Bcast.c -o Ex4_Bcast.x

Executar:

\$ mpirun -n 8 Ex4_Bcast.x

```
int main(int argc, char const *argv[])
{
    int r, p, valor, mestre = 0;

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    // Cada processo tem um valor inicial diferente
    valor = r * 10;

    // cada rank imprime seu valor antes do Bcast:
    printf("O processo %d possui valor: %d\n", r, valor);

    if (r==mestre)
    {
        printf("Entre um valor: \n");
        scanf("%d", &valor);
    }

    // A rotina de difusão é chamada por todos processos
    MPI_Bcast(&valor, 1, MPI_INT, mestre, MPI_COMM_WORLD);

    // O valor agora é o mesmo em todos os processos
    printf("O processo %d recebeu o valor: %d\n", r, valor);

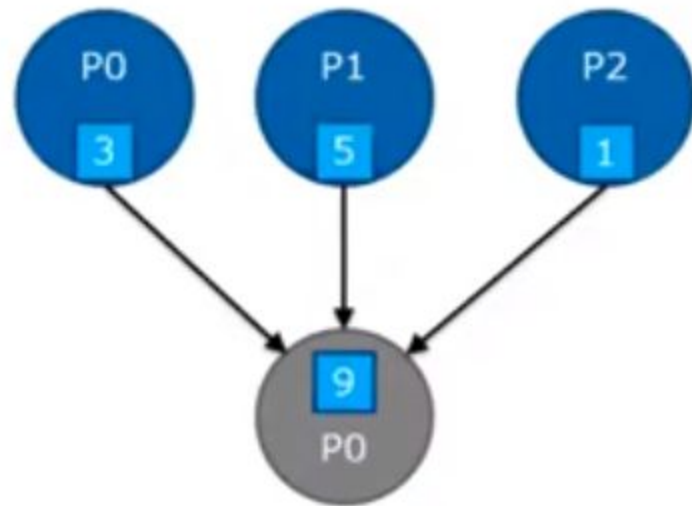
    MPI_Finalize();
    return 0;
}
```

Comunicação Coletiva: Reduction

C binding

```
int MPI_Reduce(const void *sendbuf, void *recvbuf, int count,  
              MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
```

- Cada rank envia seu dado para um único rank
- E efetua uma certa operação:
 - soma
 - min
 - max
 - ... muitas!

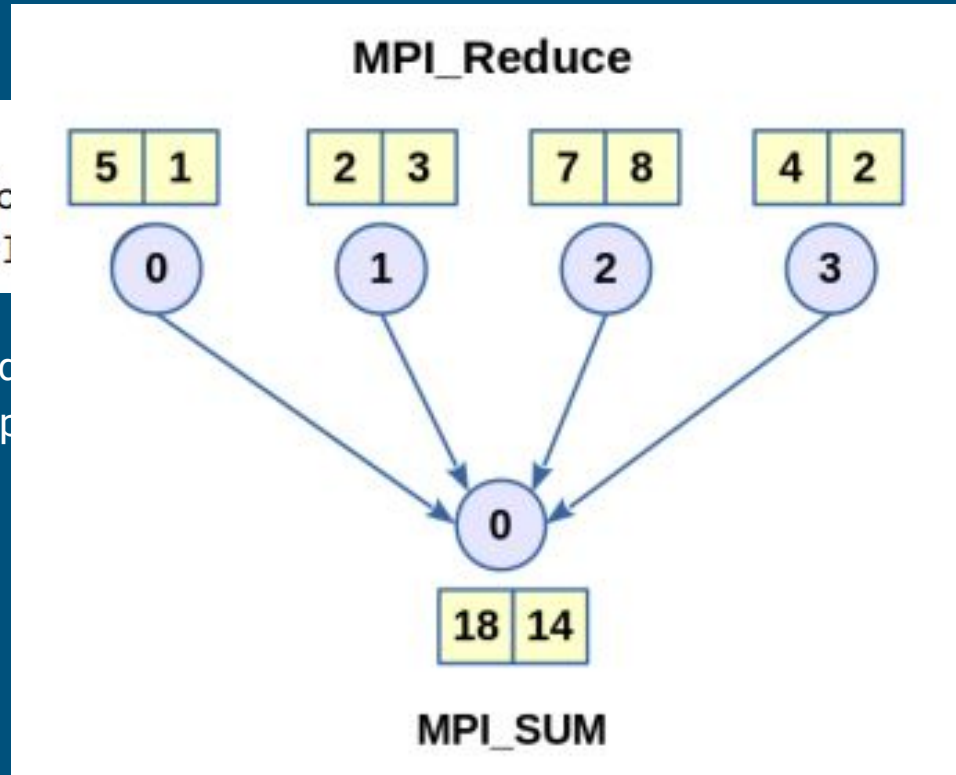


Comunicação Coletiva: Reduction

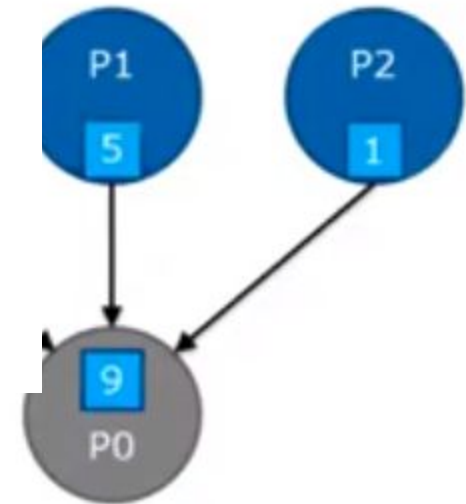
C binding

```
int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

- Cada rank envia seu dado
- E efetua uma certa operação
 - soma
 - min
 - max
 - ... muitas!



```
int, MPI_Comm comm)
```



Exemplo **Reduction**

```
$ cd Mini_Curso_MPI/Ex4_CommColetiva/mpi/Reduction
```

Compilar:

```
$ mpicc Ex4_Red.c -o Ex4_Red.x
```

Executar:

```
$ mpirun -n 4 Ex4_Red.x
```

```
int main(int argc, char *argv[])
{
    int r, p, i, mestre = 0;
    float vet_envia [TAM] ; /* Vetor a ser enviado */
    float vet_recebe [TAM]; /* Vetor a ser recebido */

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    /* Preenche o vetor com valores que dependem do ranque */
    for (i = 0; i < TAM; i++)
    {
        vet_envia[i] = (float) (r*TAM+i);
        vet_recebe[i] = 0.0;
    }

    /* Faz a redução, encontrando o valor máximo do vetor */
    MPI_Reduce(vet_envia, vet_recebe, TAM, MPI_FLOAT, MPI_MAX, mestre, MPI_COMM_WORLD);

    /* O processo raiz imprime o resultado da redução */
    if (r == mestre)
    {
        for (i = 0; i < TAM; i++)
        {
            printf("vet_recebe[%d] = %3.1f ", i, vet_recebe[i]);
            printf("\n\n");
        }
    }

    MPI_Finalize();
    return(0);
}
```

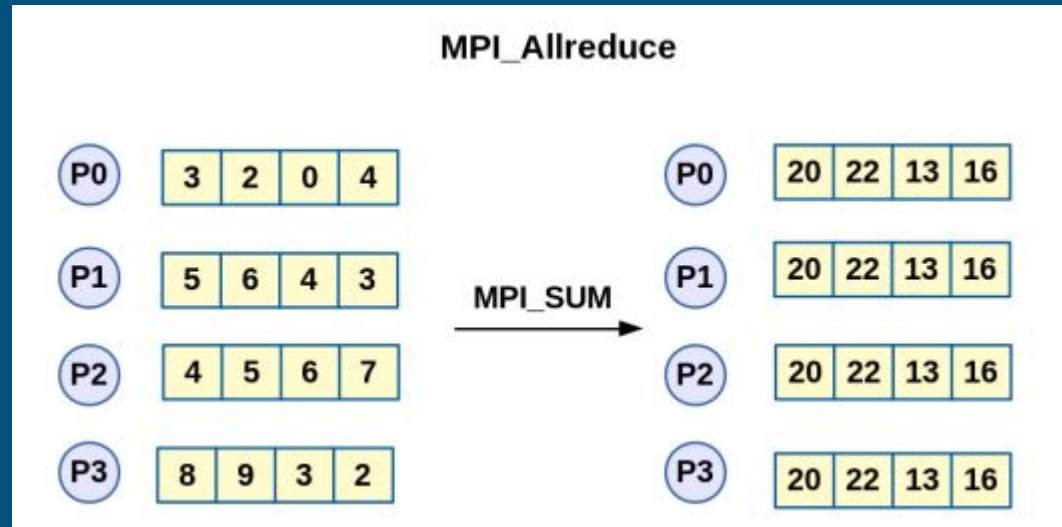
Comunicação Coletiva: AllReduction

C binding

```
int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count,  
                 MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

Seus parâmetros são similares aos da função MPI_Reduce;

A diferença é de que não existe o parâmetro “rank mestre”, já que o resultado da operação “op” será recebido por todos os processos do comunicador

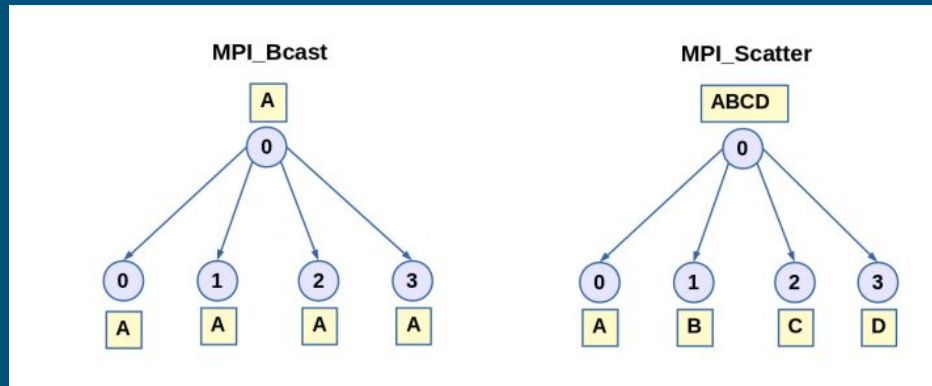


Comunicação Coletiva: Scatter

C binding

```
int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype,  
               void *recvbuf, int recvcount, MPI_Datatype recvtype, int root,  
               MPI_Comm comm)
```

- Permite enviar partes distintas de um vetor de dados do processo raiz (mestre) para todos os processos do grupo;



Exemplo Scatter

```
$ cd Mini_Curso_MPI/Ex4_CommColetiva/mpi/Scatter
```

Compilar:

```
$ mpicc Ex4_Scatter.c -o Ex4_Scatter.x
```

Executar:

```
$ mpirun -n 8 Ex4_Scatter.x
```

```
int main(int argc, char const *argv[])
{
    int r, p, valor, mestre = 0;
    int *vet_envia, vet_recebe[TAM_VET];

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    // o mestre aloca o espaço de memória e inicia o veto
    if (r == mestre)
    {
        vet_envia = (int*) malloc (p*TAM_VET*sizeof(int));
        for (int i = 0; i < (TAM_VET*p); i++)
        {
            vet_envia[i] = i*10;
            printf("r %d, vet_envia[%d]: %d\n", r, i, vet_envia[i]);
        }
        printf("\n");
    }

    // O vetor é distribuído em partes iguais entre todos ranks
    MPI_Scatter(vet_envia, TAM_VET, MPI_INT, vet_recebe, TAM_VET, MPI_INT, mestre, MPI_COMM_WORLD);

    // Cada processo imprime a parte que recebeu */
    for (int i = 0; i < TAM_VET; i++)
    {
        printf("Processo = %d, vet_recebe[%d]: %d\n", r, i, vet_recebe[i]);
    }
    printf("\n");

    MPI_Finalize();
    return 0;
}
```

1o Exemplo numérico!!!

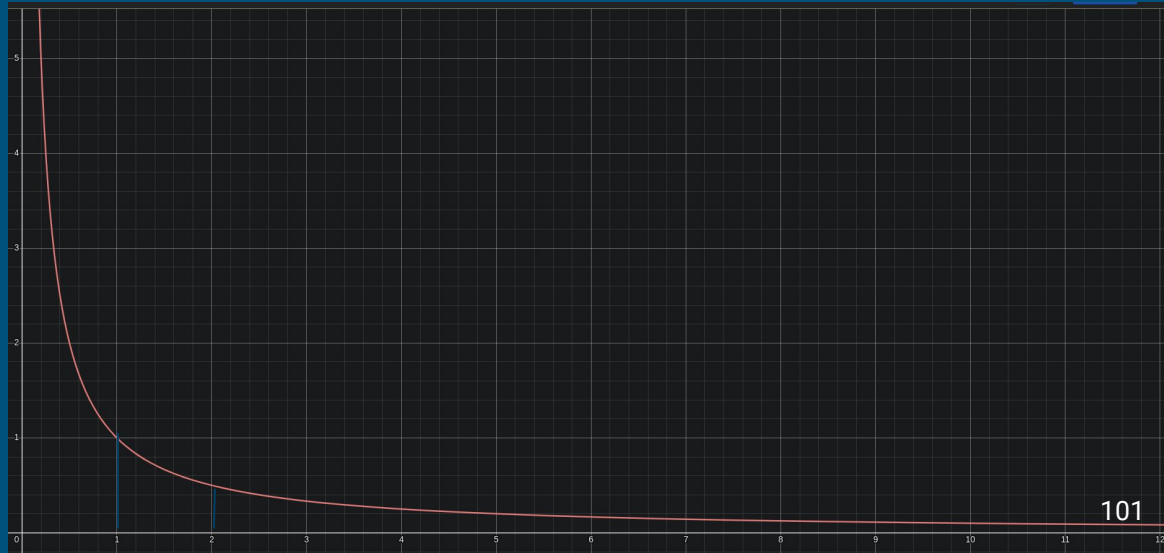
Dúvidas?



1o Exemplo numérico

Cálculo de uma integral pelo Método dos Trapézios

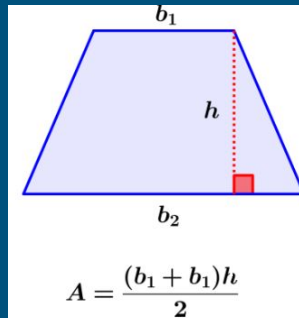
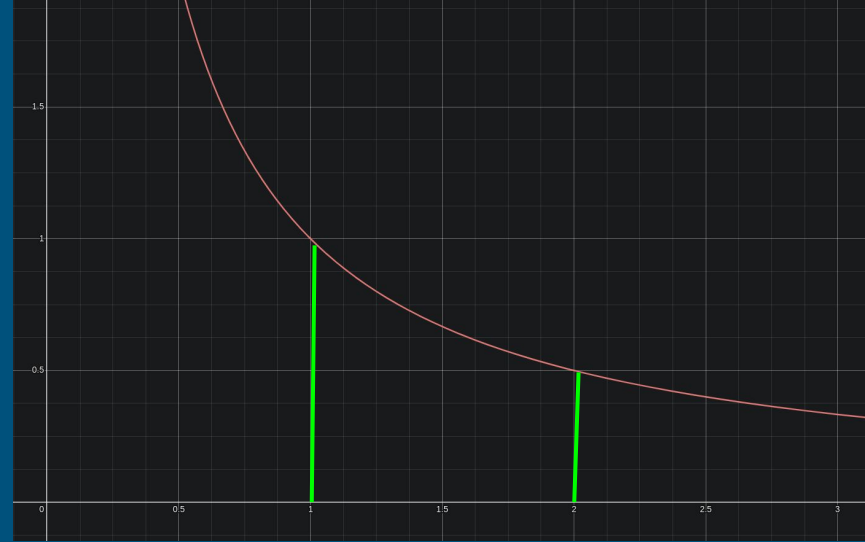
- Dada uma função não-negativa $f(x) = 1/x$
- Calcular a integral definida de $f(x)$ num intervalo $[a,b]$
- $a=1$
- $b=2$
- Com 5 subintervalos
- Depois com 10 subintervalos



1o Exemplo numérico

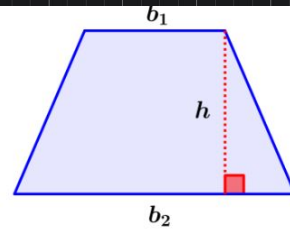
Cálculo de uma integral pelo Método dos Trapézios

- Dada uma função não-negativa $f(x) = 1/x$
- Calcular a integral definida de $f(x)$ num intervalo $[a,b]$
- $a=1$
- $b=2$
- Com 5 subintervalos
- Depois com 10 subintervalos



1o Exemplo numérico

$$h = \frac{b-a}{n} = \frac{2-1}{5} = 0.2$$



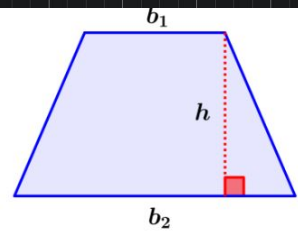
$$A = \frac{(b_1 + b_2)h}{2}$$

1o Exemplo numérico

$$h = \frac{b-a}{n} = \frac{2-1}{5} = 0.2$$

Tabela de pontos:

| | x | $f(x) = 1/x$ | |
|-------|-----|--------------|-------|
| x_0 | 1.0 | 1.0 | y_0 |
| x_1 | 1.2 | 0.833 | y_1 |
| x_2 | 1.4 | 0.714 | y_2 |
| x_3 | 1.6 | 0.625 | y_3 |
| x_4 | 1.8 | 0.556 | y_4 |
| x_5 | 2.0 | 0.500 | y_5 |



$$A = \frac{(b_1 + b_2)h}{2}$$

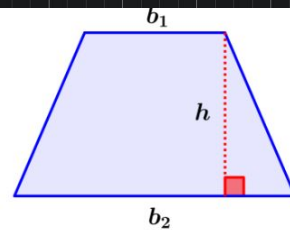
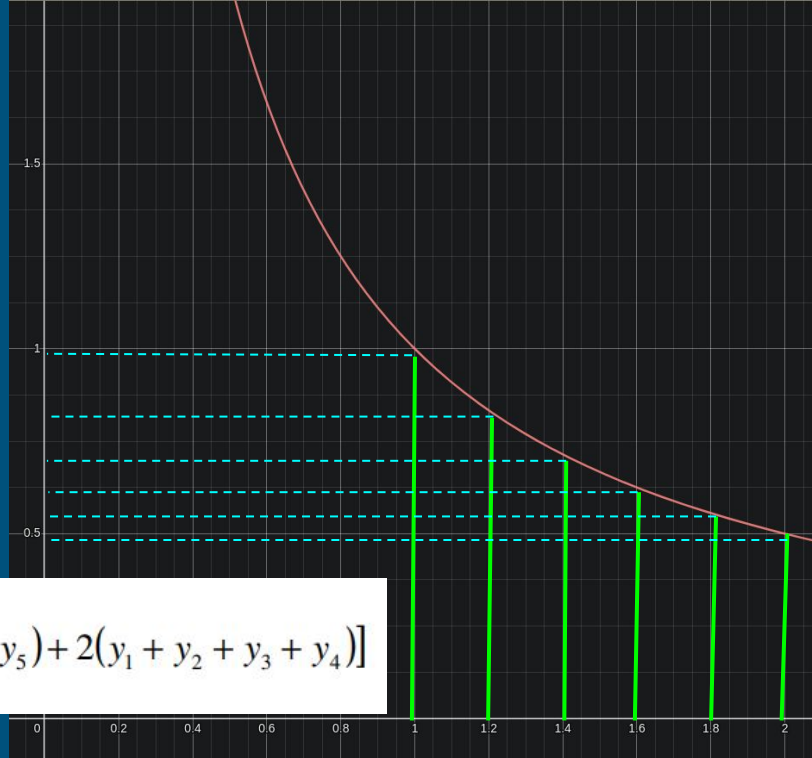
1o Exemplo numérico

$$h = \frac{b-a}{n} = \frac{2-1}{5} = 0.2$$

Tabela de pontos:

| | x | $f(x) = 1/x$ | |
|-------|-----|--------------|-------|
| x_0 | 1.0 | 1.0 | y_0 |
| x_1 | 1.2 | 0.833 | y_1 |
| x_2 | 1.4 | 0.714 | y_2 |
| x_3 | 1.6 | 0.625 | y_3 |
| x_4 | 1.8 | 0.556 | y_4 |
| x_5 | 2.0 | 0.500 | y_5 |

$$\therefore \int_1^2 \frac{1}{x} dx \cong \frac{h}{2}(SE + 2SM) = \frac{h}{2}[(y_0 + y_5) + 2(y_1 + y_2 + y_3 + y_4)]$$



$$A = \frac{(b_1 + b_2)h}{2}$$

1o Exemplo numérico

$$h = \frac{b-a}{n} = \frac{2-1}{5} = 0.2$$

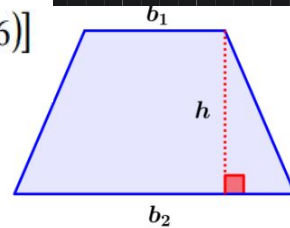
Tabela de pontos:

| | x | $f(x) = 1/x$ | |
|-------|-----|--------------|-------|
| x_0 | 1.0 | 1.0 | y_0 |
| x_1 | 1.2 | 0.833 | y_1 |
| x_2 | 1.4 | 0.714 | y_2 |
| x_3 | 1.6 | 0.625 | y_3 |
| x_4 | 1.8 | 0.556 | y_4 |
| x_5 | 2.0 | 0.500 | y_5 |

$$\therefore \int_1^2 \frac{1}{x} dx \cong \frac{h}{2} (SE + 2SM) = \frac{h}{2} [(y_0 + y_5) + 2(y_1 + y_2 + y_3 + y_4)]$$

$$\therefore \int_1^2 \frac{1}{x} dx \cong \frac{h}{2} (SE + 2SM) = \frac{0.2}{2} [(1.0 + 0.5) + 2(0.833 + 0.714 + 0.625 + 0.556)]$$

$$= 0.1(1.5 + (2)(2.728)) = 0.6956$$



$$A = \frac{(b_1 + b_2)h}{2}$$

1o Exemplo numérico

$$h = \frac{b-a}{n} = \frac{2-1}{10} = 0.1$$

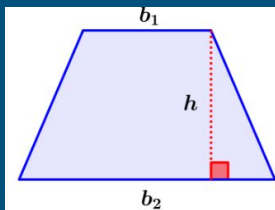
Tabela de pontos:

| x | | | | f(x) = 1/x | | | |
|-------|-----|------------|-------|------------|-----|------------|----------|
| x | | f(x) = 1/x | | x | | f(x) = 1/x | |
| x_0 | 1.0 | 1.0 | y_0 | x_6 | 1.6 | 0.6250 | y_6 |
| x_1 | 1.1 | 0.9091 | y_1 | x_7 | 1.7 | 0.5882 | y_7 |
| x_2 | 1.2 | 0.8333 | y_2 | x_8 | 1.8 | 0.5556 | y_8 |
| x_3 | 1.3 | 0.7692 | y_3 | x_9 | 1.9 | 0.5263 | y_9 |
| x_4 | 1.4 | 0.7143 | y_4 | x_{10} | 2.0 | 0.5000 | y_{10} |
| x_5 | 1.5 | 0.6667 | y_5 | | | | |

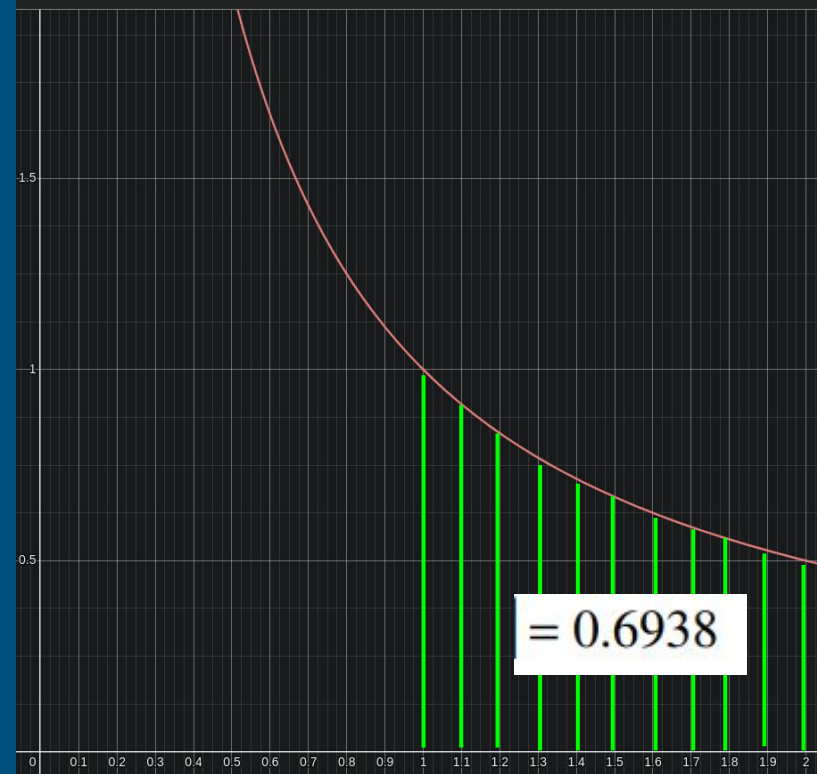
$$SE = 1 + 0.5 = 1.5$$

$$SM = 0.9091 + 0.8333 + 0.7692 + 0.7143 + 0.6667 + 0.6250 + 0.5882 + 0.5556 + 0.5263 = 6.1877$$

$$\therefore \int_1^2 \frac{1}{x} dx \cong \frac{h}{2} (SE + 2SM) = \frac{0.1}{2} [1.5 + 2(6.1877)] = 0.6938$$



$$A = \frac{(b_1 + b_2)h}{2}$$



(c) *Cálculo analítico:*

$$\int_1^2 \frac{1}{x} dx = [\ln x]_1^2 = \ln 2 - \ln 1 = 0.6931 - 0 = 0.6931$$

$$M = \max_{1 \leq x \leq 2} |f''(x)| = \frac{2}{1^3} = 2$$

1o Exemplo numérico

Cálculo de uma integral pelo Método dos Trapézios

- Dada uma função não-negativa $f(x) = 1/x$
- Calcular a integral definida de $f(x)$ num intervalo $[a,b] :: [1,2]$

Solução:

- Dividir o intervalo $[a,b]$ em N partes: tal que a largura de cada intervalo $h = (b-a)/N$
- Aproximar a área de cada intervalo com a área do trapézio de cada intervalo
- i -ésimo intervalo = $[x_{i-1}, x_{i-1} + h] = [x_{i-1}, x_i]$
- Área do i -ésimo intervalo $\approx [f(x_{i-1}) + f(x_i)] \cdot h / 2$
- Área total $[f(x_0)/2 + f(x_n)/2 + f(x_1) + f(x_2) + \dots + f(x_{n-1})] \cdot h$

1o Exemplo numérico

Cálculo de uma integral pelo Método dos Trapézios

- Dada uma função não-negativa $f(x) = 1/x$
- Calcular a integral definida de $f(x)$ num intervalo

Solução:

- Dividir o intervalo $[a,b]$ em N partes: tal que
- Aproximar a área de cada intervalo com a
- i -ésimo intervalo = $[x_{i-1}, x_{i-1} + h] = [x_{i-1}, x_i]$
- Área do i -ésimo intervalo $\approx [f(x_{i-1}) + f(x_i)] \cdot h$
- Área total $[f(x_0)/2 + f(x_n)/2 + f(x_1) + f(x_2) + \dots + f(x_{n-1})]$

```
int main(int argc, char const *argv[])
{
    double a, b, h, x, integral;
    long int n, i;

    a = 1.0;
    b = 2.0;
    n = 10;
    h = (b - a) / n;

    integral = (f(a) + f(b)) / 2.0;
    x = a;

    for (i = 1; i <= n-1; i++){
        x = x + h;
        integral = integral + f(x);
    }
    integral = integral * h;

    printf("\n Valor da integral definida eh: %6.4f\n ", integral);
    return 0;
}

float f(float x)
{
    float resultado;
    resultado = 1.0 / x;
    return(resultado);
}
```

1o Exemplo numérico

Cálculo de uma integral pelo Método dos Trapézios

- Dada uma função não-negativa $f(x) = 1/x$
- Calcular a integral definida de $f(x)$ num intervalo $[a,b] :: [1,2]$
- Medir o tempo de execução do programa:
 - `time executavel.x`
 - $n = 10, 1000, 1\ 000\ 000, 1\ 000\ 000\ 000, \dots$

1o Exemplo numérico

Cálculo de uma integral pelo Método dos Trapézios

Estratégia paralela:

- Dividir os N intervalos entre os P processadores
 - Cada rank calcula a integral da sua parte
 - Cada rank envia seu resultado para o mestre (P_0)
 - P_0 soma tudo e imprime o resultado final
-
- Resolver:
 - a, b, n para cada rank
 - Determinar o subdomínio de trabalho de cada rank
 - P_0 calcula tudo e envia para os ranks
 - Cada rank calcula seu próprio subdomínio

1o Exemplo numérico

Cálculo de uma integral pelo Método dos Trapézios

Estratégia paralela:

- Dividir o N intervalos entre os P processadores
- Cada rank calcula a integral da sua parte

```
double trap(double local_a, double local_b, long int local_n, double h)
{
    long int i;
    double integral, x;

    integral = (f(local_a) + f(local_b)) / 2.0;
    x = local_a;

    for (i = 1; i <= local_n-1; i++){
        x = x + h;
        integral = integral + f(x);
    }
    integral = integral * h;

    return(integral);
}
```

a rank

1o Exemplo numérico

Cálculo de uma integral pelo Método dos Trapézios

Estratégia paralela:

- Dividir o N intervalos entre os P processadores
- Cada rank calcula a integral da sua parte

```
double trap(double local_a, double local_b, long int local_n, long int r)
{
    long int i;
    double integral, x;

    integral = (f(local_a) + f(local_b)) / 2.0;
    x = local_a;

    for (i = 1; i <= local_n-1; i++){
        x = x + h;
        integral = integral + f(x);
    }
    integral = integral * h;

    return(integral);
}
```

```
int main(int argc, char const *argv[])
{
    double a, b, h, integral, local_a, local_b, total;
    long int n, i, local_n;
    int r, p, tag = 1, source, dest = 0;

    MPI_Status status;

    a = 1.0;
    b = 2.0;
    n = 10;
    h = (b - a) / n;

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    local_n = n / p; // qtde de passos para cada rank
    local_a = r * local_n * h + a; // início do intervalo de cada rank
    local_b = local_a + local_n * h; // fim do intervalo de cada rank

    // cada rank calcula a integral de sua parte:
    integral = trap(local_a, local_b, local_n, h);

    if ( r == 0 ){ // rank 0 recebe o resultado de cada rank e soma tudo:
        total = integral;
        for ( source = 1; source < p; source ++){
            MPI_Recv(&integral, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
            total = total + integral;
        }
    } else { // outros ranks enviam seu resultado para o rank 0:
        MPI_Send(&integral, 1, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
    }

    // rank 0 imprime o resultado final:
    if ( r == 0 ) printf("\n Valor da integral definida eh: %.4f\n ", total);

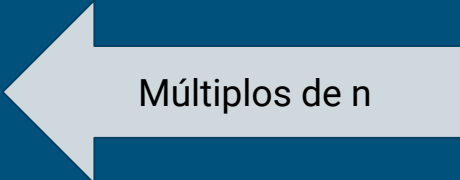
    MPI_Finalize();
    return 0;
}
```

1o Exemplo numérico

Cálculo de uma integral pelo Método dos Trapézios

Calculando o desempenho paralelo:

- Executar a versão sequencial:
 - para $n = 10\,000\,000\,000$
 - `time Ex5_Trap_mpi.x`
 - anotar o tempo!
- Executar a versão paralela para o mesmo caso:
 - $n = 10\,000\,000\,000$
 - `time mpirun -n p Ex5_Trap_mpi.x`
 - Variando **p** de 2, 4, 8, ... até quantos cores?
 - `cat /proc/cpuinfo`
 - anotar os tempos! [média de 3 execuções]



Múltiplos de n

1o Exemplo numérico

Mini_Curso_MPI/Ex4_MetTrap/mpi

Cálculo de uma integral pelo Método dos Trapézios

Calculando o desempenho paralelo:

- Executar a versão sequencial:
 - para $n = 10\,000\,000\,000$
 - `time Ex5_Trap_mpi.x`
 - anotar o tempo!
- Executar a versão paralela para o mesmo caso:
 - $n = 10\,000\,000\,000$
 - `time mpirun -n p Ex5_Trap_mpi.x`
 - Variando **p** de 2, 4, 8, ... até quantos cores?
 - `cat /proc/cpuinfo`
 - anotar os tempos! [média de 3 execuções]

Como medir, mensurar o ganho?

Múltiplos de n

1o Exemplo numérico

Mini_Curso_MPI/Ex4_MetTrap/mpi

Cálculo de uma integral pelo Método dos Trapézios

Calculando o desempenho paralelo:

- Executar a versão sequencial:
 - para $n = 10\,000\,000\,000$
 - `time Ex5_Trap_mpi.x`
 - anotar o tempo!
- Executar a versão paralela para o mesmo caso:
 - $n = 10\,000\,000\,000$
 - `time mpirun -n p Ex5_Trap_mpi.x`
 - Variando **p** de 2, 4, 8, ... até quantos cores?
 - `cat /proc/cpuinfo`
 - anotar os tempos! [média de 3 execuções]

Como medir, mensurar o ganho?

2 medidas básicas de desempenho paralelo:

- 1- Speedup**
- 2- Eficiência!**

Cálculo de desempenho paralelo

Speedup:

O speedup (ou aceleração) é a relação entre o tempo gasto para executar uma tarefa com um único processo e o tempo gasto com p processos. Quanto mais próximo de p for o speedup, melhor, pois se aproxima do speedup linear (p), considerado ideal

Então o Speedup é calculado dessa maneira:

$$S_p = \frac{t_1}{t_p}$$

Onde:

S_p : *speedup* utilizando p processos;

t_1 : tempo sequencial;

t_p : tempo utilizando p processos.

Cálculo de desempenho paralelo

Eficiência:

A eficiência é o speedup normalizado pelo número de processos, ou seja, quanto mais próximo de um (ou 100%) melhor

A eficiência é definida por:

$$E_p = \frac{S_p}{p}$$

Onde:

E_p : eficiência utilizando p processos;

S_p : *speedup* utilizando p processos;

p : número de processos.

Cálculo de desempenho paralelo

Interpretação das métricas:

Ideal:

- Utilizando o **dobro** de processadores na versão paralela, obter a **metade** do tempo do sequencial;
- $p=2, 4, 8$
- $S_p = 2x, 4x, 8x$
- $E_p = 1$
- Mas nem sempre acontece assim!
- Existem partes do programa que não são paralelizáveis, ou seja, sempre serão executadas de forma sequencial: (Lei de Amdhal e de Gustavson)
 - Entrada de dados
 - Impressão de resultados
 - Qualquer computação que exija cálculos lineares ou com dependência de dados

Cálculo de desempenho paralelo

$$S_p = \frac{t_1}{t_p}$$

Plotando os resultados:

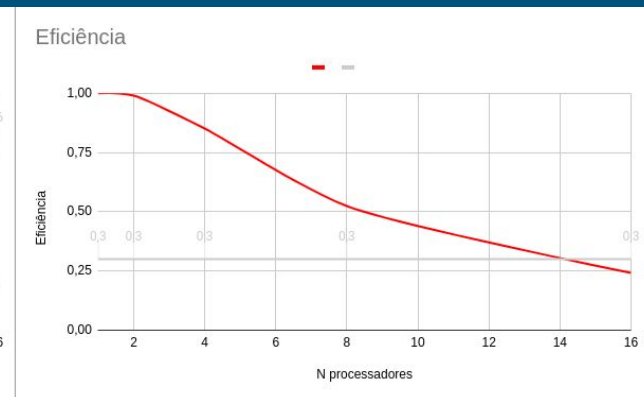
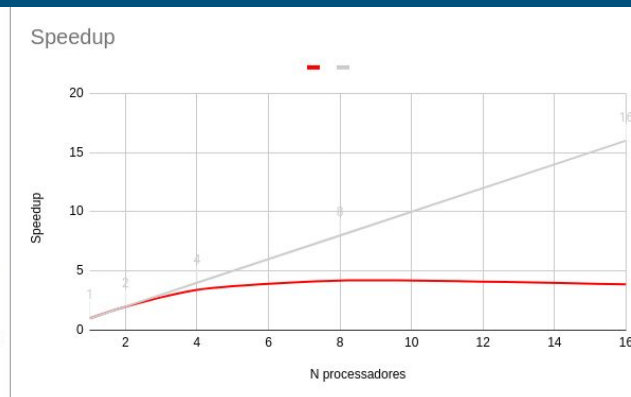
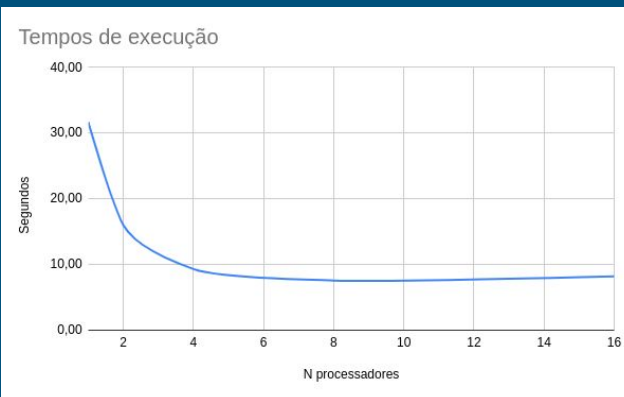
Inicialmente, fazemos uma tabela:

$$E_p = \frac{S_p}{p}$$

| Num. de Ranks: | 1 | 2 | 4 | 8 | 16 |
|----------------|-------|-------|------|------|------|
| Tempos (s): | 31,66 | 15,98 | 9,30 | 7,55 | 8,19 |
| Speedup: | 1,00 | 1,98 | 3,40 | 4,19 | 3,87 |
| Eficiencia: | 1,00 | 0,99 | 0,85 | 0,52 | 0,24 |

Cálculo de desempenho paralelo

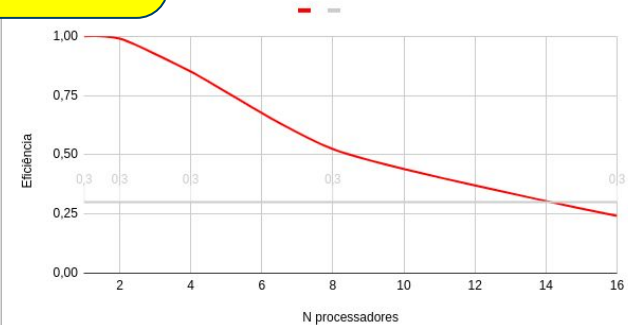
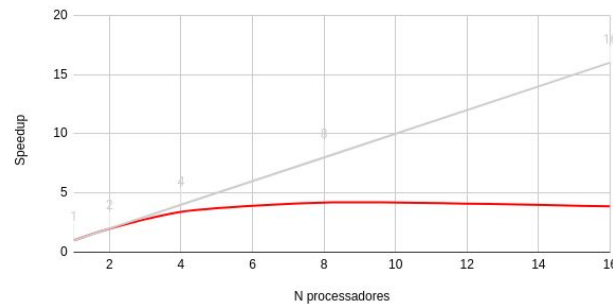
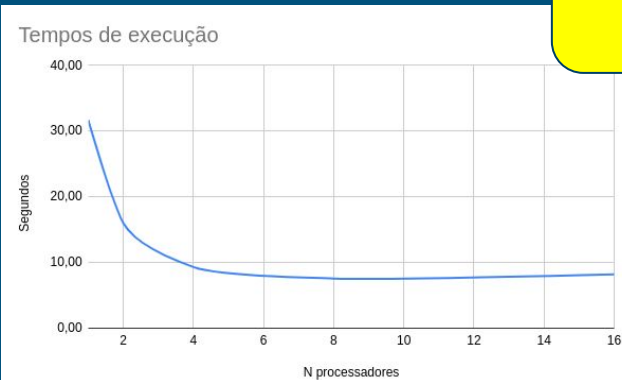
| Num. de Ranks: | 1 | 2 | 4 | 8 | 16 |
|----------------|-------|-------|------|------|------|
| Tempos (s): | 31,66 | 15,98 | 9,30 | 7,55 | 8,19 |
| Speedup: | 1,00 | 1,98 | 3,40 | 4,19 | 3,87 |
| Eficiencia: | 1,00 | 0,99 | 0,85 | 0,52 | 0,24 |



Cálculo de desempenho paralelo

| Num. de Ranks: | 1 | 2 | 4 | 8 | 16 |
|----------------|-------|-------|------|------|------|
| Tempos (s): | 31,66 | 15,98 | 9,30 | 7,55 | 8,19 |
| Speedup: | 1,00 | 1,98 | 3,40 | 4,19 | 3,87 |
| Eficiencia: | 1,00 | 0,99 | 0,85 | 0,52 | 0,24 |

Como melhorar meu desempenho?



Como melhorar meu desempenho?

Cálculo de uma integral pelo Método dos Trapézios

Estratégia paralela:

- Dividir o N intervalos entre os P processadores
- Cada rank calcula a integral da sua parte

```
double trap(double local_a, double local_b, long int local_n, double f_a, double f_b)
{
    long int i;
    double integral, x;

    integral = (f_a + f_b) / 2.0;
    x = local_a;

    for (i = 1; i <= local_n-1; i++){
        x = x + h;
        integral = integral + f(x);
    }
    integral = integral * h;

    return(integral);
}
```

```
int main(int argc, char const *argv[])
{
    double a, b, h, integral, local_a, local_b, total;
    long int n, i, local_n;
    int r, p, tag = 1, source, dest = 0;

    MPI_Status status;

    a = 1.0;
    b = 2.0;
    n = 10;
    h = (b - a) / n;

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    local_n = n / p; // qtde de passos para cada rank
    local_a = r * local_n * h + a; // início do intervalo de cada rank
    local_b = local_a + local_n * h; // fim do intervalo de cada rank

    // cada rank calcula a integral de sua parte:
    integral = trap(local_a, local_b, local_n, f_a, f_b);

    if ( r == 0 ){ // rank 0 recebe o resultado de cada rank e soma tudo:
        total = integral;
        for ( source = 1; source < p; source ++ ){
            MPI_Recv(&integral, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
            total = total + integral;
        }
    } else { // outros ranks enviam seu resultado para o rank 0:
        MPI_Send(&integral, 1, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
    }

    // rank 0 imprime o resultado final:
    if ( r == 0 ) printf("\n Valor da integral definida eh: %.4f\n ", total);

    MPI_Finalize();
    return 0;
}
```



```

int main(int argc, char const *argv[])
{
    int r, p, i, mestre = 0;
    float vet_envia [TAM]; /* Vetor a ser enviado */
    float vet_recebe [TAM]; /* Vetor a ser recebido */

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    /* Preenche o vetor com valores que dependem do ranque */
    for (i = 0; i < TAM; i++)
    {
        vet_envia[i] = (float) (r*TAM+i);
        vet_recebe[i] = 0.0;
    }

    /* Faz a redução, encontrando o valor máximo do vetor */
    MPI_Reduce(vet_envia, vet_recebe, TAM, MPI_FLOAT, MPI_MAX, mestre, MPI_COMM_WORLD);

    /* O processo raiz imprime o resultado da redução */
    if (r == mestre)
    {
        for (i = 0; i < TAM; i++)
            printf("vet_recebe[%d] = %3.1f ", i, vet_recebe[i]);
        printf("\n\n");
    }
}

```

```

MPI_Finalize();
return(0);
}

```

```

integral =
x = local_a

```

```

for (i = 1;
    x = x +
    integral
}

```

```

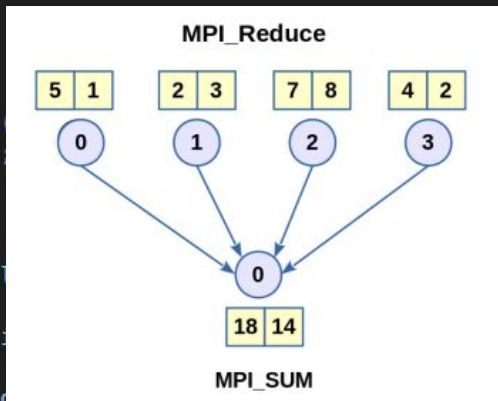
integral =

```

```

return(integral);
}

```



```

int main(int argc, char const *argv[])
{
    double a, b, h, integral, local_a, local_b, total;
    long int n, i, local_n;
    int r, p, tag = 1, source, dest = 0;

    MPI_Status status;

    a = 1.0;
    b = 2.0;
    n = 10;
    h = (b - a) / n;

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    local_n = n / p; // qtde de passos para cada rank
    local_a = r * local_n * h + a; // início do intervalo de cada rank
    local_b = local_a + local_n * h; // fim do intervalo de cada rank

    // cada rank calcula a integral de sua parte:
    integral = trap(local_a, local_b, local_n, h);

    if ( r == 0 ){ // rank 0 recebe o resultado de cada rank e soma tudo:
        total = integral;
        for ( source = 1; source < p; source ++){
            MPI_Recv(&integral, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &status);
            total = total + integral;
        }
    } else { // outros ranks enviam seu resultado para o rank 0:
        MPI_Send(&integral, 1, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
    }

    // rank 0 imprime o resultado final:
    if ( r == 0 ) printf("\n Valor da integral definida eh: %6.4f\n ", total);

    MPI_Finalize();
    return 0;
}

```

```

int main(int argc, char argv[])
{
    int r, p, i, mestre = 0;
    float vet_envia [TAM] ; /* Vetor a ser enviado */
    float vet_recebe [TAM]; /* Vetor a ser recebido */

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    /* Preenche o vetor com valores que dependem do ranque */
    for (i = 0; i < TAM; i++)
    {
        vet_envia[i] = (float) (r*TAM+i);
        vet_recebe[i] = 0.0;
    }
}

```

```

/* Faz a redução, encontrando o valor máximo do vetor */
MPI_Reduce(vet_envia, vet_recebe, TAM, MPI_FLOAT, MPI_MAX, mestre, MPI_COMM_WORLD);

```

```

/* O processo raiz imprime o resultado da redução */
if (r == mestre)
{
    for (i = 0; i < TAM; i++)
        printf("vet_recebe[%d] = %3.1f ", i, vet_recebe[i]);
    printf("\n\n");
}

```

```

MPI_Finalize();
return(0);
}

```

```

integral =
x = local_a

```

```

for (i = 1;
    x = x +
    integral
}

```

```

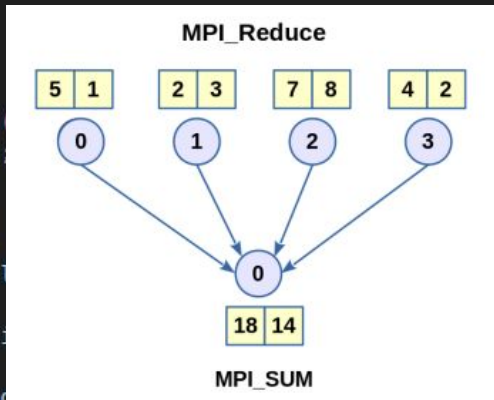
integral =

```

```

return(integral);
}

```



```

int main(int argc, char const *argv[])
{
    double a, b, h, integral, local_a, local_b, total;
    long int n, i, local_n;
    int r, p, tag = 1, source, dest = 0;

    MPI_Status status;

    a = 1.0;
    b = 2.0;
    n = 10;
    h = (b - a) / n;

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &r);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    local_n = n / p; // qtde de passos para cada rank
    local_a = r * local_n * h + a; // início do intervalo de cada rank
    local_b = local_a + local_n * h; // fim do intervalo de cada rank

    // cada rank calcula a integral de sua parte:
    integral = trap(local_a, local_b, local_n, h);

    if ( r == 0 ){ // rank 0 recebe o resultado de cada rank e soma tudo:
        total = integral;
        for ( source = 1; source < p; source ++){
            MPI_Recv(&integral, 1, MPI_DOUBLE, source , tag, MPI_COMM_WORLD, &status);
            total = total + integral;
        }
    } else { // outros ranks enviam seu resultado para o rank 0:
        MPI_Send(&integral, 1, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
    }

    // rank 0 imprime o resultado final:
    if ( r == 0 ) printf("\n Valor da integral definida eh: %6.4f\n ", total);

    MPI_Finalize();
    return 0;
}

```

Modificar e medir o desempenho!

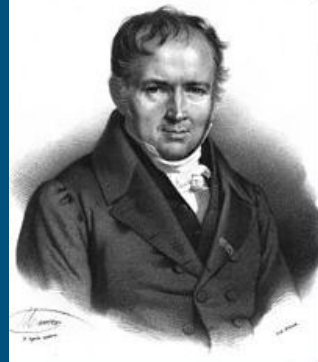
Agenda

1. Introdução à programação paralela
2. MPI
3. Estrutura de um programa paralelo
4. Principais funções MPI
5. Exemplos iniciais
6. Análise desempenho
7. A equação do calor de Poisson
8. Modelagem do problema
9. Solução numérica sequencial
10. Solução numérica paralela MPI + Análise desempenho
11. Introdução ao MPI - Shared Memory
12. Solução numérica paralela MPI-SHM + Análise desempenho

Algoritmo da solução:

1. Entender o mecanismos do problema;
2. Discretizar o problema:
 - a. Natureza é contínua
 - b. Computação é discreta
3. Modelar seu problema num sistema simplificado e controlado;
4. Executar sequencial, tempo base para cálculo de desempenho paralelo;
5. Paralelizar seu modelo;
6. Conferir resultados: versão paralela == sequencial
7. Executar versão paralela com vários cores, tomada de tempo!
8. Cálculo de desempenho

A Equação do calor de Poisson



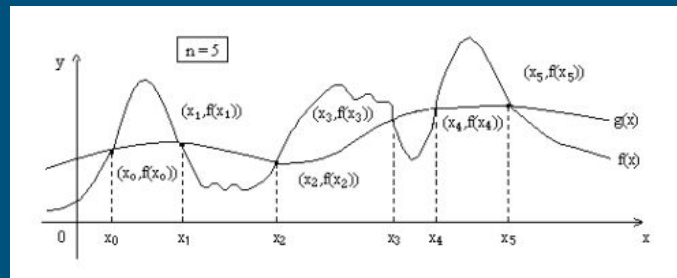
- Equação Diferencial Parcial
- Com solução muito usada no eletromagnetismo, matemática, física aplicada...
- Inclusive em modelagem de previsão de tempo e clima!
- Derivadas de segunda ordem em duas direções, 2D
- Usa Laplaciano, Equação de Laplace

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

Métodos Numéricos

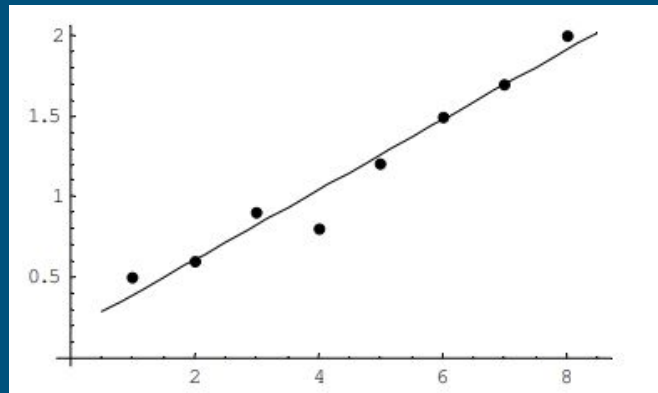
- São métodos (computacionais) que resolvem problemas (matemáticos) que não possuem soluções analíticas;
- Muitas áreas da matemática e da física aplicada:
- Integração numérica:
 - Quando a primitiva não é conhecida ou não é de fácil obtenção;
 - Quando a função a ser integrada é conhecida apenas em alguns pontos (empírica);
- Interpolações! Muito útil
 - Quando a função não é conhecida e deseja obter o valor em determinados pontos
 - 1D, 2D, e até 3D

| T (°C) | 25 | 30 | 35 | 40 |
|--------|---------|---------|---------|---------|
| c | 0.99852 | 0.99826 | 0.99818 | 0.99828 |



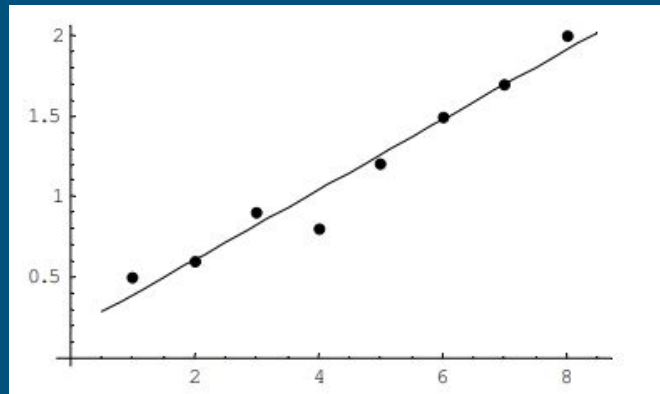
Métodos Numéricos

- Integração numérica:
- Interpolações numéricas
- Equações algébricas e transcendentais
 - Equações trigonométricas, Exponenciais,...
- Sistemas de Equações Lineares
 - Matrizes esparsas
 - Métodos iterativos e diretos
- Ajustes de curvas
 - Mínimos quadrados, de vários graus
- EDOs
 - Runge-Kutta's (1a, ..., 4a ordem)
- EPDs
 - Diferenças Finitas
 - Volumes Finitos
 - Muitos métodos!



Métodos Numéricos

- Integração numérica:
- Interpolações numéricas
- Equações algébricas e transcendentais
 - Equações trigonométricas, Exponenciais,...
- Sistemas de Equações Lineares
 - Matrizes esparsas
 - Métodos iterativos e diretos
- Ajustes de curvas
 - Mínimos quadrados, de vários graus
- EDOs
 - Runge-Kutta's (1a, ..., 4a ordem)
- EPDs
 - **Diferenças Finitas**
 - Volumes Finitos
 - Muitos métodos!



Algoritmo da solução:

- ~~1. Entender o mecanismos do problema;~~
2. Discretizar o problema:
 - a. Natureza é contínua
 - b. Computação é discreta
3. Modelar seu problema num sistema simplificado e controlado;
4. Executar sequencial, tempo base para cálculo de desempenho paralelo;
5. Paralelizar seu modelo;
6. Conferir resultados: versão paralela == sequencial
7. Executar versão paralela com vários cores, tomada de tempo!
8. Cálculo de desempenho

Métodos Numéricos - Diferenças Finitas

- Aproximar as derivadas das ED pelo quociente das diferenças
- Obtida da Série de Taylor
- Discretizando as equações

Dif.Fin. Progressiva

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

Dif.Fin. Regressiva

$$f'(x) = \frac{f(x) - f(x-h)}{h}$$

Dif.Fin. Centrada

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

Dif.Fin. de segunda ordem centrada

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Pré modelando o problema

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y)$$

Pré modelando o problema

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y)$$



$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Pré modelando o problema

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y)$$



$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

$$\frac{\partial^2 U}{\partial x^2} \approx \frac{U(x+h, y) - 2U(x, y) + U(x-h, y)}{h^2}$$

Pré modelando o problema

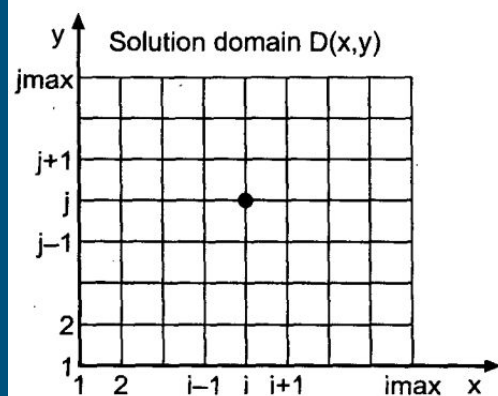
$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y)$$



$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

$$\frac{\partial^2 U}{\partial x^2} \approx \frac{U(x+h, y) - 2U(x, y) + U(x-h, y)}{h^2}$$

$$U(x, y) = U_{i,j}, U(x+h, y) = U_{i+1,j},$$



Pré modelando o problema

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y)$$

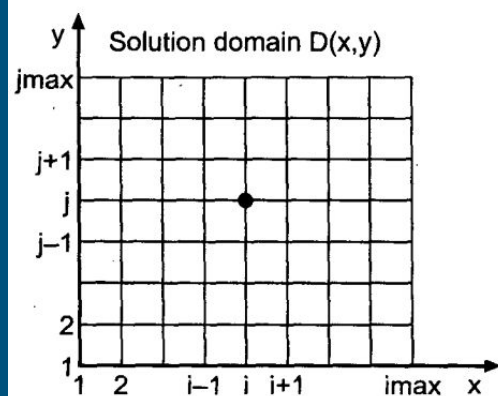


$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

$$\frac{\partial^2 U}{\partial x^2} \approx \frac{U(x+h, y) - 2U(x, y) + U(x-h, y)}{h^2}$$

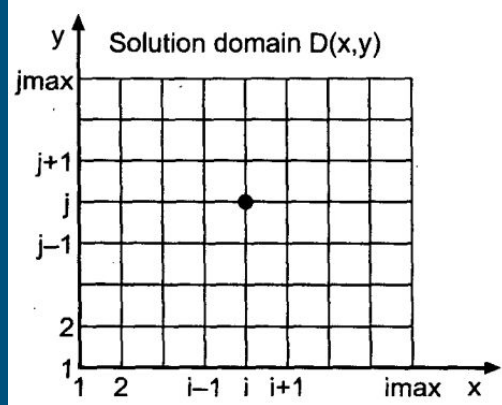
$$U(x, y) = U_{i,j}, U(x+h, y) = U_{i+1,j},$$

$$\frac{\partial^2 U}{\partial x^2} \approx \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{(\Delta x)^2}$$



Pré modelando o problema

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y)$$



$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \approx \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{(\Delta x)^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{(\Delta y)^2}$$

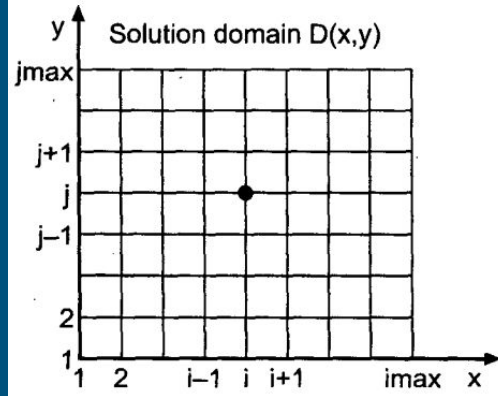
$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \approx \frac{U_{i+1,j} + U_{i,j+1} - 4U_{i,j} + U_{i-1,j} + U_{i,j-1}}{h^2}$$

Pré modelando o problema

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y)$$

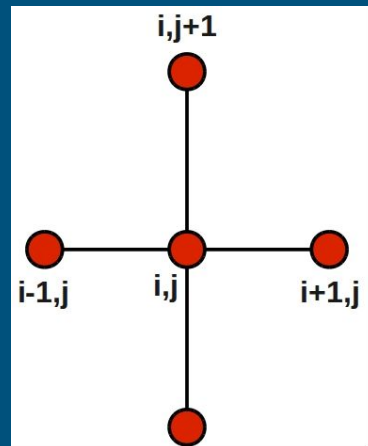
Stencil 5 pontos!

P_{ij} = média dos pontos vizinhos!



$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \approx \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{(\Delta x)^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{(\Delta y)^2}$$

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \approx \frac{U_{i+1,j} + U_{i,j+1} - 4U_{i,j} + U_{i-1,j} + U_{i,j-1}}{h^2}$$

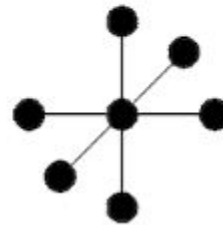
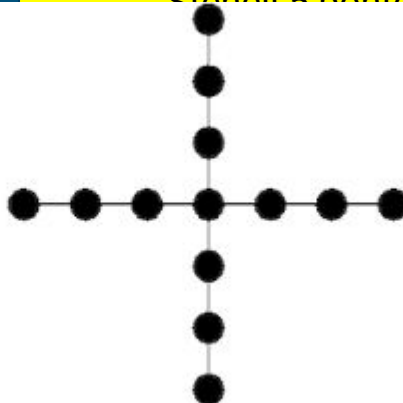
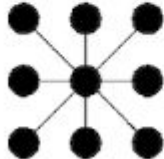
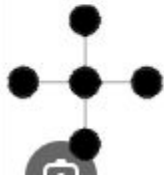


Pré modelando o problema

$$\frac{\partial^2 U}{\partial x^2}$$

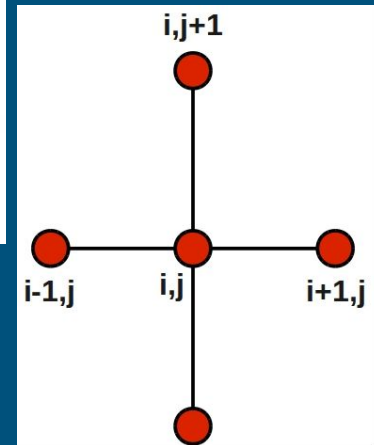
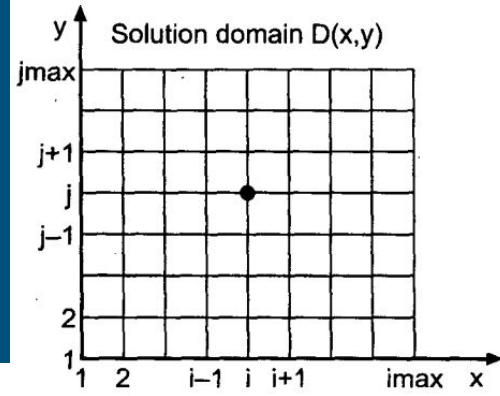
$$\frac{\partial^2 U}{\partial y^2}$$

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2}$$



Stencil 5-point

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \approx \frac{U_{i+1,j} + U_{i,j+1} - 4U_{i,j} + U_{i-1,j} + U_{i,j-1}}{h^2}$$



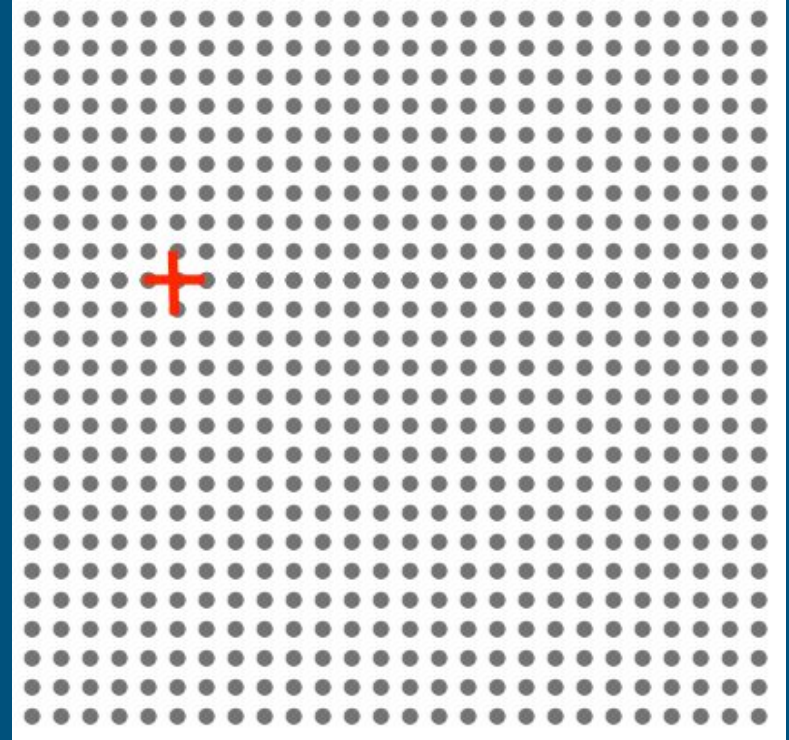
Algoritmo da solução:

- ~~1. Entender o mecanismos do problema;~~
- ~~2. Discretizar o problema:~~
 - Natureza é contínua
 - Computação é discreta
3. Modelar seu problema num sistema simplificado e controlado;
4. Executar sequencial, tempo base para cálculo de desempenho paralelo;
5. Paralelizar seu modelo;
6. Conferir resultados: versão paralela == sequencial
7. Executar versão paralela com vários cores, tomada de tempo!
8. Cálculo de desempenho

Pré modelando o problema - Estratégia sequencial

1. Passo:

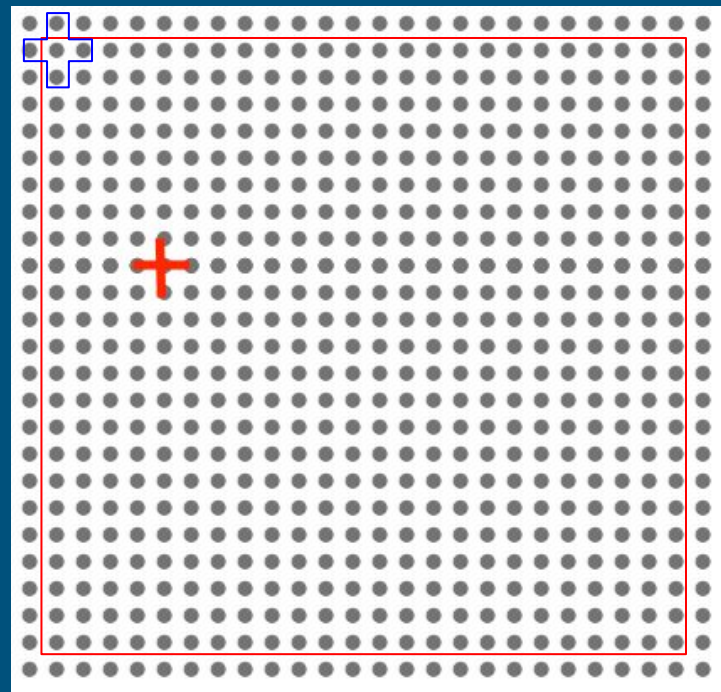
- Discretizar nosso domínio, a placa;
- Equação diferencial será calculada em cada ponto
- Stencil 5 pontos está marcado na cruz vermelha;
- O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- Mas e para computar os pontos das bordas?



Pré modelando o problema - Estratégia sequencial

1. Passo:

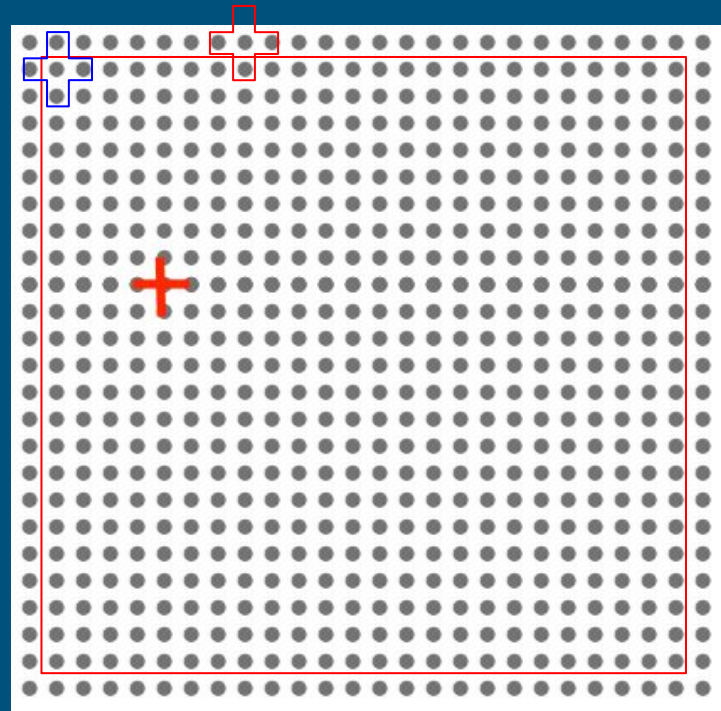
- Discretizar nosso domínio, a placa;
- Equação diferencial será calculada em cada ponto
- Stencil 5 pontos está marcado na cruz vermelha;
- O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- Mas e para computar os pontos das bordas?



Pré modelando o problema - Estratégia sequencial

1. Passo:

- Discretizar nosso domínio, a placa;
- Equação diferencial será calculada em cada ponto
- Stencil 5 pontos está marcado na cruz vermelha;
- O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- Mas e para computar os pontos das bordas?

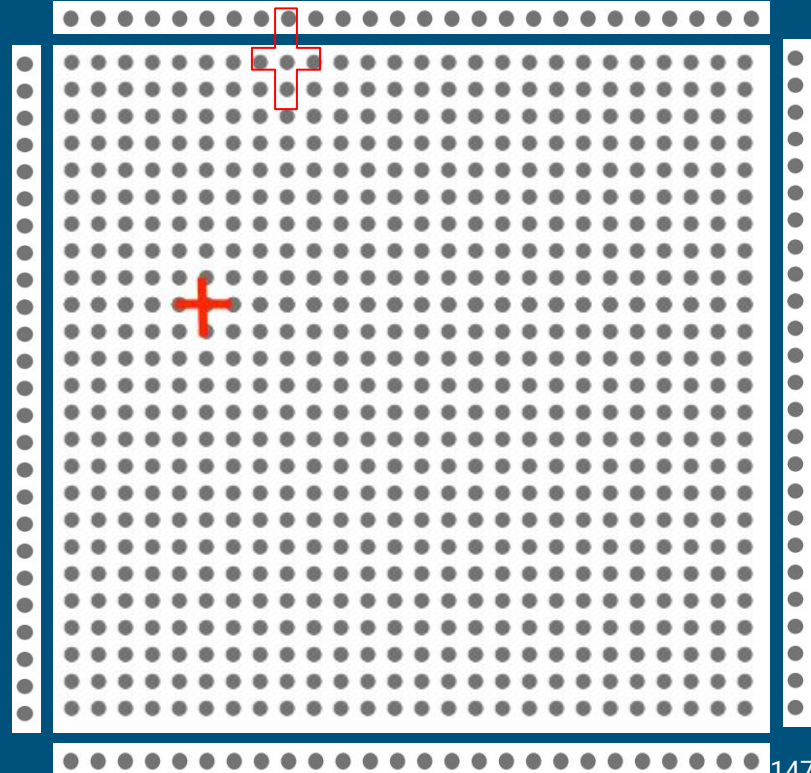


Pré modelando o problema - Estratégia sequencial

1. Passo:

- Discretizar nosso domínio, a placa;
- Equação diferencial será calculada em cada ponto
- Stencil 5 pontos está marcado na cruz vermelha;
- O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- Mas e para computar os pontos das bordas?

Criar uma borda extra!



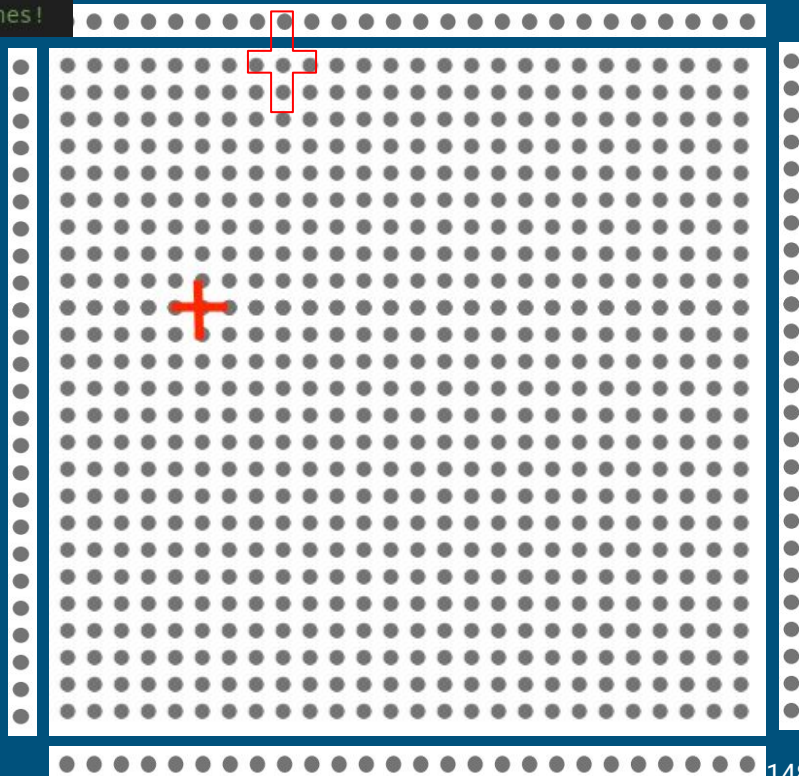
Pré modelando o problema - Estratégia sequencial

```
double *aold = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo zones!  
double *anew = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo-zones!
```

1. Passo:

- Discretizar nosso domínio, a placa;
- Equação diferencial será calculada em cada ponto
- Stencil 5 pontos está marcado na cruz vermelha;
- O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- Mas e para computar os pontos das bordas?

Criar uma borda extra!



Pré modelando o problema - Estratégia sequencial

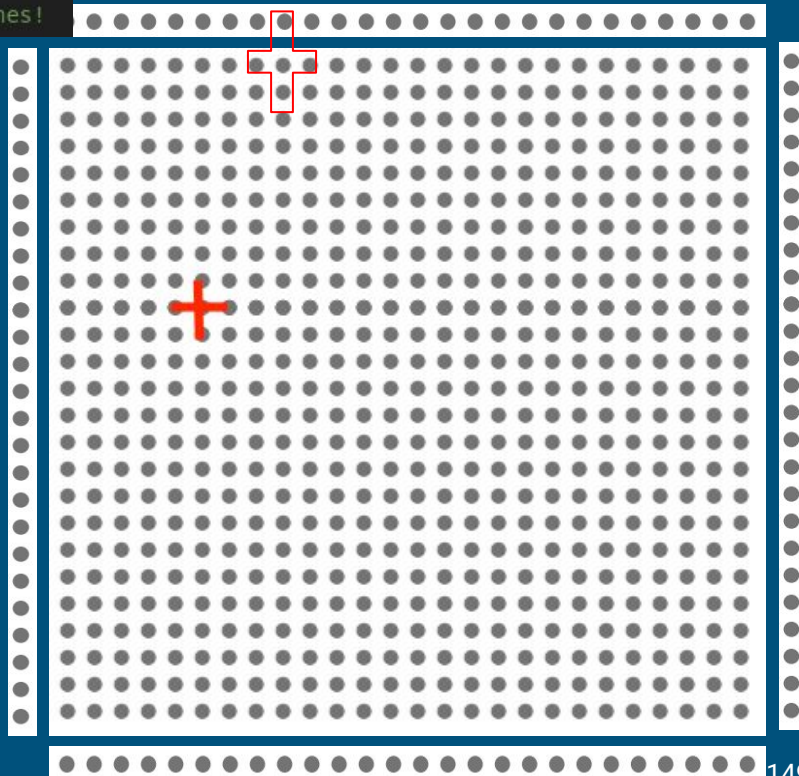
```
double *aold = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo zones!  
double *anew = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo-zones!
```

```
// row-major order  
#define ind(i,j) (j)*(n+2)+i
```

domínio, a placa:
ial será calculada em cada

- c. Stencil 5 pontos está marcado na cruz vermelha;
- d. O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- e. $A_u = f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- f. Mas e para computar os pontos das bordas?

Criar uma borda extra!



Pré modelando o problema - Estratégia sequencial

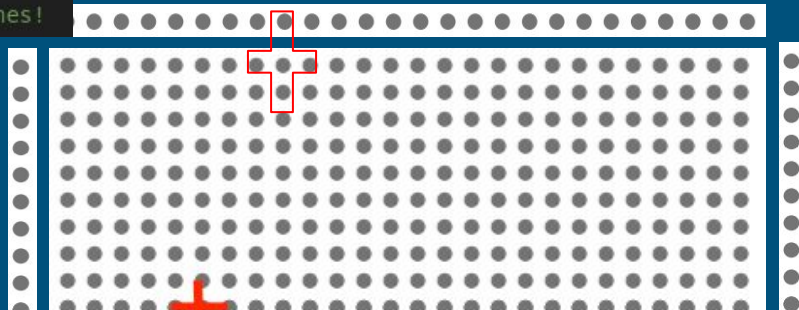
```
double *aold = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo zones!  
double *anew = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo-zones!
```

```
// row-major order  
#define ind(i,j) (j)*(n+2)+i
```

domínio, a placa:
ial será calculada em cada

ponto

c. Stencil 5 pontos está marcado na cruz



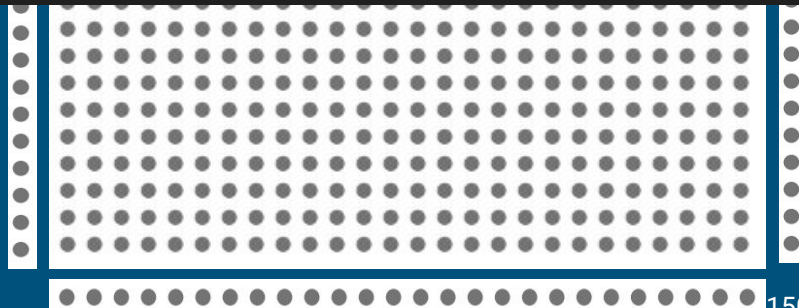
```
for(j=1; j<n+1; ++j)  
  for(i=1; i<n+1; ++i)  
    anew[ind(i,j)] = aold[ind(i,j)]/2.0 + (aold[ind(i-1,j)] + aold[ind(i+1,j)] + aold[ind(i,j-1)] + aold[ind(i,j+1)])/4.0/2.0;
```

anterior;

e. $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;

f. Mas e para computar os pontos das bordas?

Criar uma borda extra!

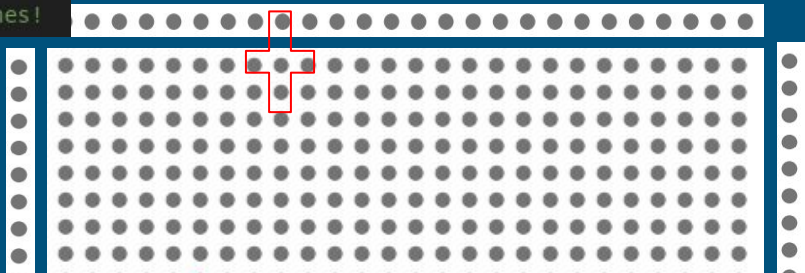


Pré modelando o problema - Estratégia sequencial

```
double *aold = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo zones!  
double *anew = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo-zones!
```

```
// row-major order  
#define ind(i,j) (j)*(n+2)+i
```

domínio, a placa:
ial será calculada em cada
ponto

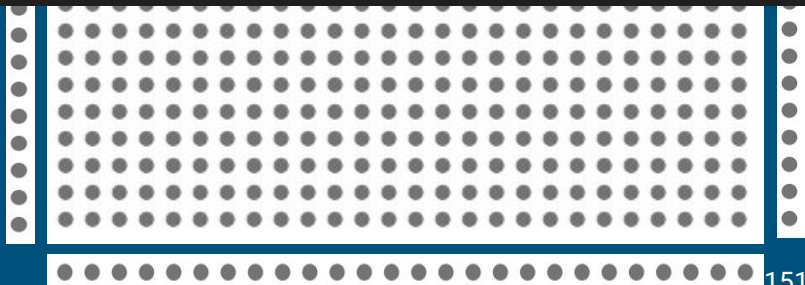


```
for(iter=0; iter<niters; iter+=2)  
{  
    for(j=1; j<n+1; ++j)  
        for(i=1; i<n+1; ++i)  
            anew[ind(i,j)] = aold[ind(i,j)]/2.0 + (aold[ind(i-1,j)] + aold[ind(i+1,j)] + aold[ind(i,j-1)] + aold[ind(i,j+1)])/4.0/2.0;
```

anterior;

- e. $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- f. Mas e para computar os pontos das bordas?

Criar uma borda extra!

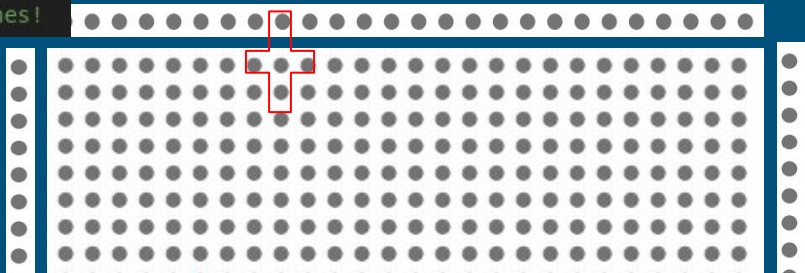


Pré modelando o problema - Estratégia sequencial

```
double *aold = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo zones!  
double *anew = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo-zones!
```

```
// row-major order  
#define ind(i,j) (j)*(n+2)+i
```

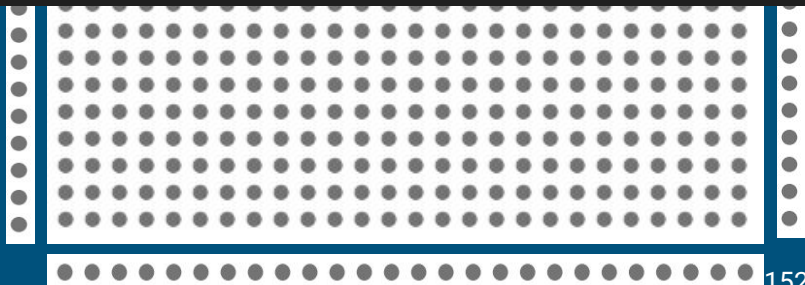
domínio, a placa:
ial será calculada em cada
ponto



```
for(iter=0; iter<niters; iter+=2)  
{  
    for(j=1; j<n+1; ++j)  
        for(i=1; i<n+1; ++i)  
            anew[ind(i,j)] = aold[ind(i,j)]/2.0 + (aold[ind(i-1,j)] + aold[ind(i+1,j)] + aold[ind(i,j-1)] + aold[ind(i,j+1)])/4.0/2.0;  
    aold=anew;  
}
```

- e. anterior, $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- f. Mas e para computar os pontos das bordas?

Criar uma borda extra!

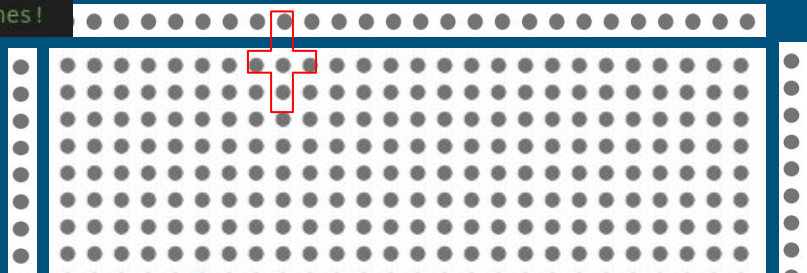


Pré modelando o problema - Estratégia sequencial

```
double *aold = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo zones!  
double *anew = (double*)calloc(1,(n+2)*(n+2)*sizeof(double)); // 1-wide halo-zones!
```

```
// row-major order  
#define ind(i,j) (j)*(n+2)+i
```

domínio, a placa:
ial será calculada em cada
ponto

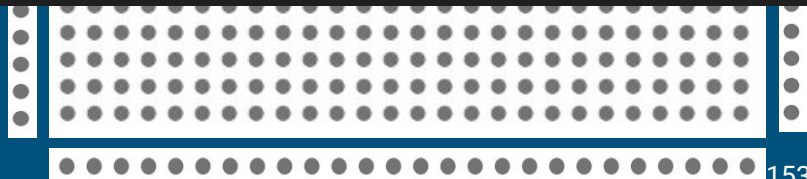


```
for(iter=0; iter<niters; iter+=2)  
{  
    for(j=1; j<n+1; ++j)  
        for(i=1; i<n+1; ++i)  
            anew[ind(i,j)] = aold[ind(i,j)]/2.0 + (aold[ind(i-1,j)] + aold[ind(i+1,j)] + aold[ind(i,j-1)] + aold[ind(i,j+1)])/4.0/2.0;  
    for(j=1; j<n+1; ++j)  
        for(i=1; i<n+1; ++i)  
            aold[ind(i,j)] = anew[ind(i,j)]/2.0 + (anew[ind(i-1,j)] + anew[ind(i+1,j)] + anew[ind(i,j-1)] + anew[ind(i,j+1)])/4.0/2.0;
```

matriz B, que atualizara a matriz A no final
da integração;

f. Mas e para computar os pontos das bordas?

Criar uma borda extra!

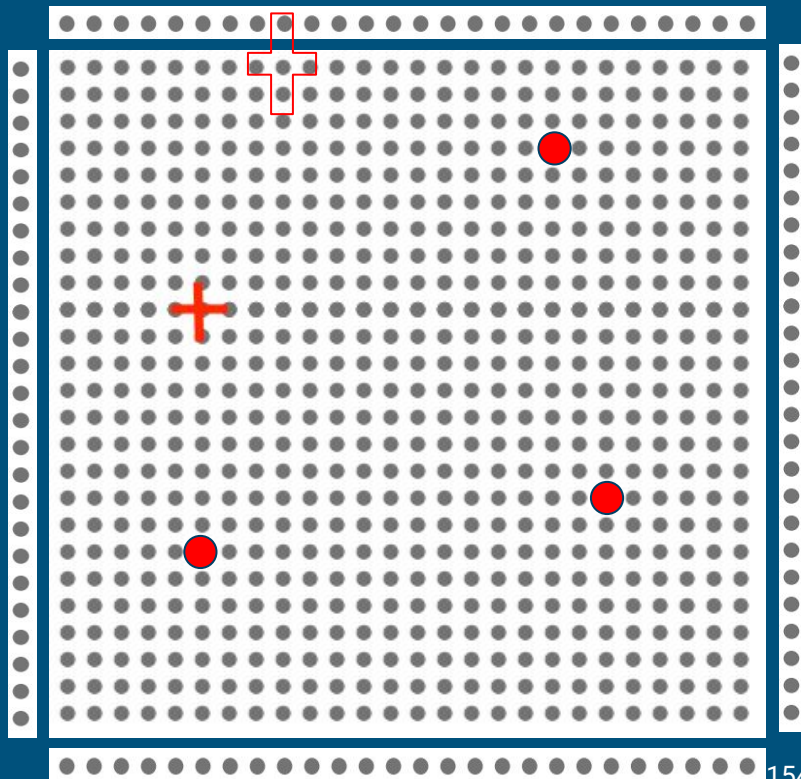


Pré modelando o problema - Estratégia sequencial

1. Passo:

- Discretizar nosso domínio, a placa;
- Equação diferencial será calculada em cada ponto
- Stencil 5 pontos está marcado na cruz vermelha;
- O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- Mas e para computar os pontos das bordas? Criar uma borda extra!

g. Adiciona-se energia ao sistema!



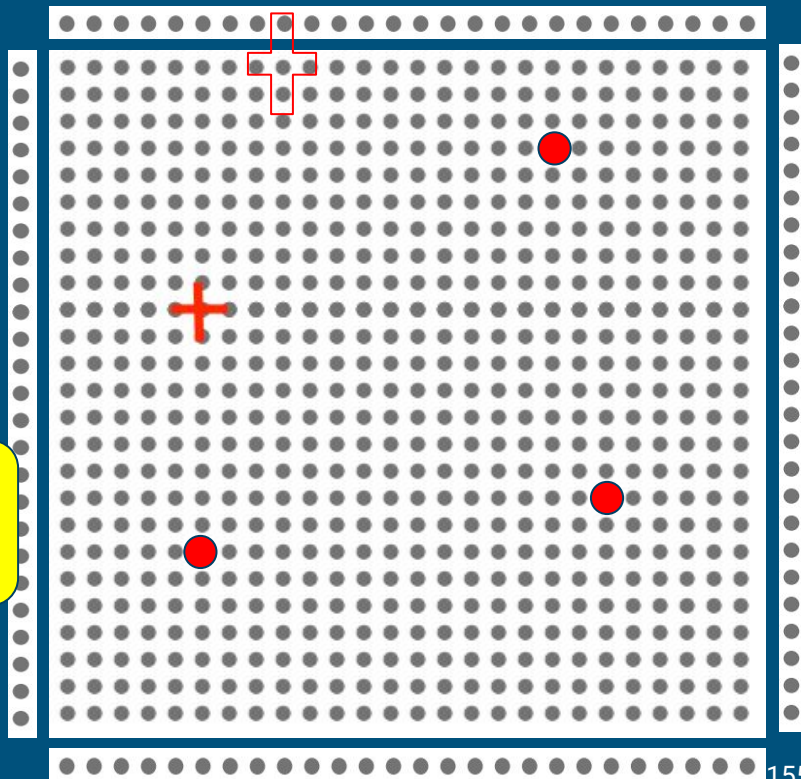
Pré modelando o problema - Estratégia sequencial

1. Passo:

- Discretizar nosso domínio, a placa;
- Equação diferencial será calculada em cada ponto
- Stencil 5 pontos está marcado na cruz vermelha;
- O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- $Au=f(x,y)$; resultado da matriz B, que atualiza a integração;
- Mas e para computar os pontos das bordas? Criar uma borda extra!

g. Adiciona-se energia ao sistema!

+1,0° C a cada
iteração (timestep)



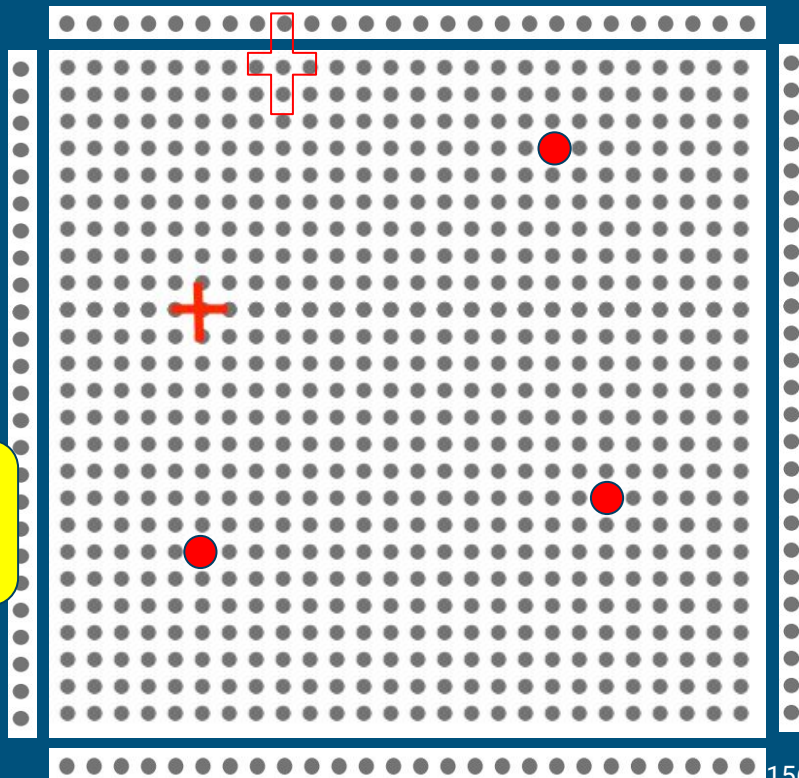
Pré modelando o problema - Estratégia sequencial

1. Passo:

- Discretizar nosso domínio, a placa;
- Equação diferencial será calculada em cada ponto
- Stencil 5 pontos está marcado na cruz vermelha;
- O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- $Au=f(x,y)$; resultado da matriz B, que atualiza a matriz A da integração;
- Mas e para computar os pontos das bordas. Criar uma borda extra!

g. Adiciona-se energia ao sistema! +3° C a cada timestep

+1,0° C a cada
iteração (timestep)



Pré modelando o problema - Estratégia sequencial

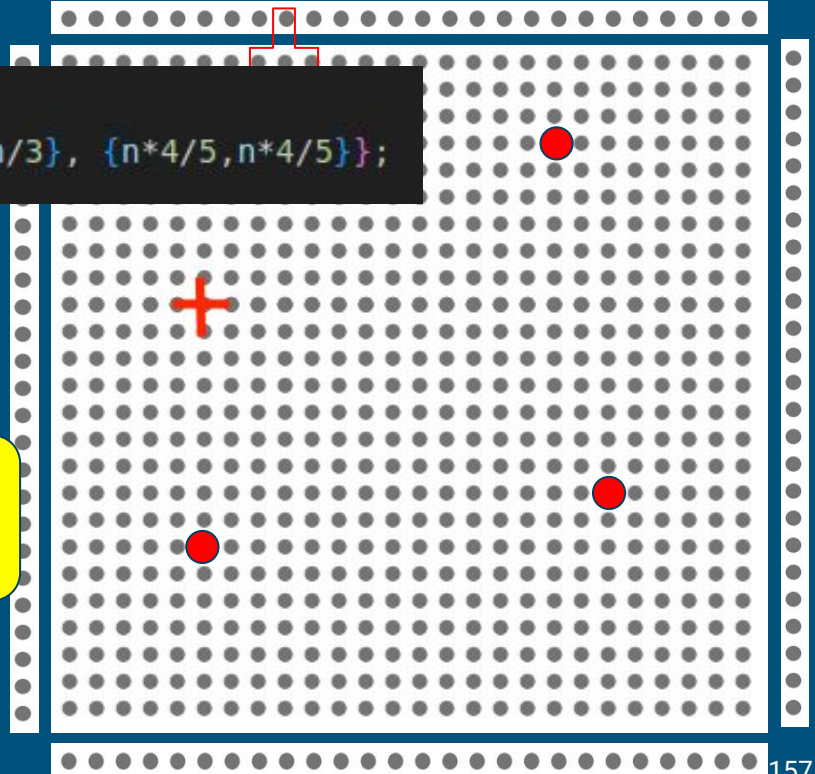
1.

```
#define nsources 3
int sources[nsources][2] = {{n/2,n/2}, {n/3,n/3}, {n*4/5,n*4/5}};
```

- c. Stencil 5 pontos está marcado na cruz vermelha;
- d. O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- e. $Au=f(x,y)$; resultado da matriz B, que atualiza a integração;
- f. Mas e para computar os pontos das bordas. Criar uma borda extra!

g. Adiciona-se energia ao sistema! +3° C a cada timestep

+1,0° C a cada
iteração (timestep)



Pré modelando o problema - Estratégia sequencial

1.

```
#define nsources 3
int sources[nsources][2] = {{n/2,n/2}, {n/3,n/3}, {n*4/5,n*4/5}};

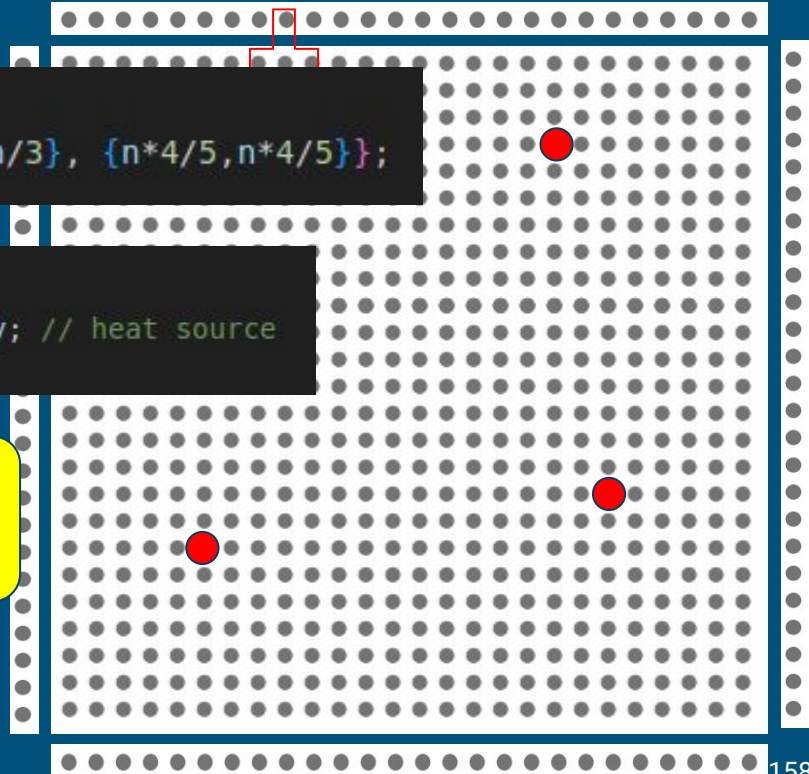
ponto
```

```
for(i=0; i<nsources; ++i)
    anew[ind(sources[i][0],sources[i][1])] += energy; // heat source
```

- e. $Au=f(x,y)$; resultado da matriz B, que atualiza a integração;
- f. Mas e para computar os pontos das bordas. Criar uma borda extra!

g. Adiciona-se energia ao sistema! +3° C a cada timestep

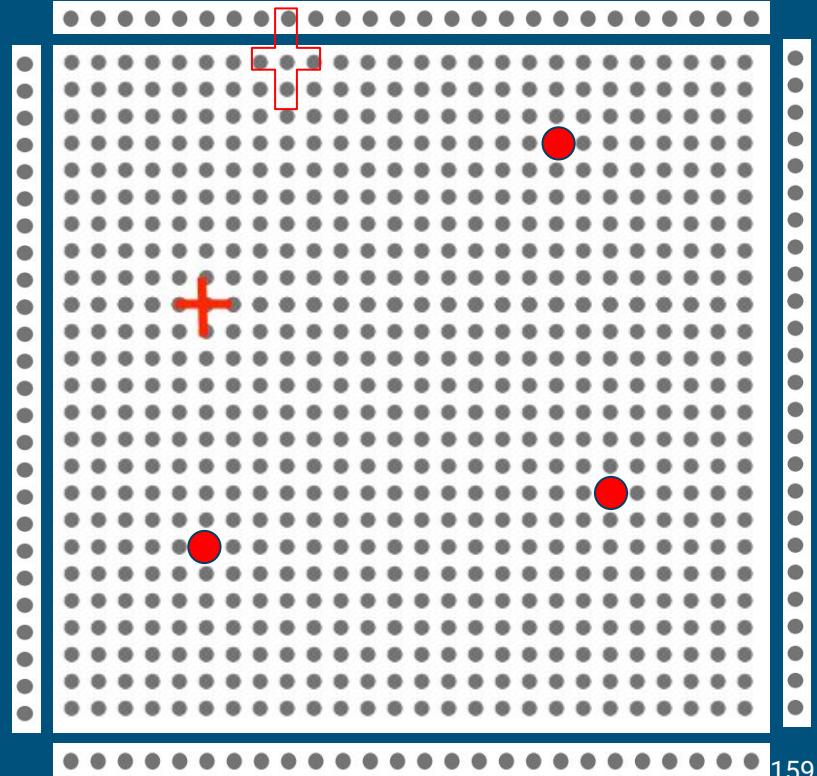
+1,0° C a cada
iteração (timestep)



Pré modelando o problema - Estratégia sequencial

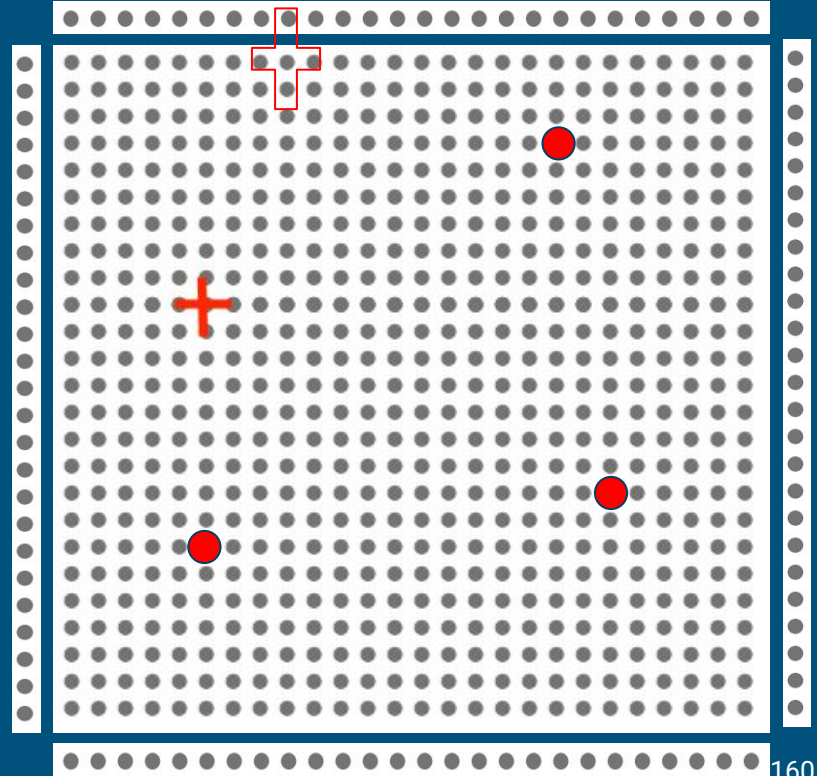
1. Passo:

- Discretizar nosso domínio, a placa:
- Equação diferencial será calculada em cada ponto
- Stencil 5 pontos está marcado na cruz vermelha;
- O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- Mas e para computar os pontos das bordas? Criar uma borda extra!
- Adiciona-se energia ao sistema! $+3^\circ\text{C}$ a cada timestep
- Como conferir se está tudo ok?**



Pré modelando o problema - Estratégia sequencial

1. Verifico a qtde total de energia ao final das integrações! a placa: calculada em cada
- c. Stencil 5 pontos está marcado na cruz vermelha;
- d. O ponto $U_{i,j}$ do próximo estado será a média dos 4 vizinhos + o valor do ponto no estado anterior;
- e. $Au=f(x,y)$; resultado armazenado em outra matriz B, que atualizará a matriz A no final da integração;
- f. Mas e para computar os pontos das bordas? Criar uma borda extra!
- g. Adiciona-se energia ao sistema! $+3^{\circ}\text{C}$ a cada timestep
- h. **Como conferir se está tudo ok?**



Pré modelando o problema - Estratégia sequencial

1.

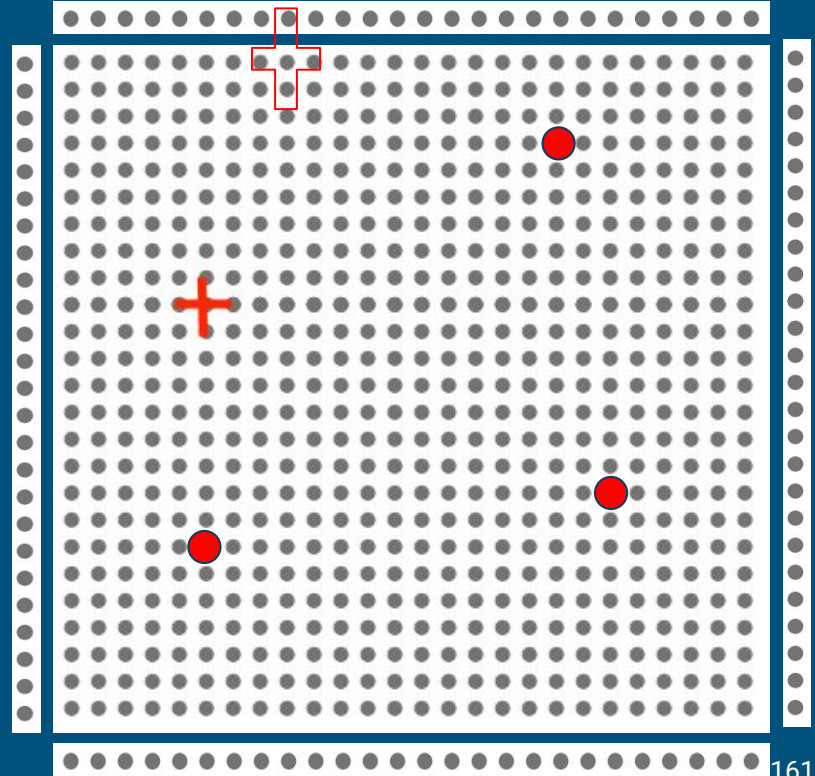
Verifico a qtde total de energia ao final das integrações!

- c. Stencil 5 pontos está marcado na cruz vermelha:

```
heat=0.0;
for(j=1; j<n+1; ++j)
  for(i=1; i<n+1; ++i)
    heat += aold[ind(i,j)];
```

estado será a média do ponto no estado armazenado em outra matriz A no final da integração;

- f. Mas e para computar os pontos das bordas? Criar uma borda extra!
- g. Adiciona-se energia ao sistema! $+3^{\circ}\text{C}$ a cada timestep
- h. Como conferir se está tudo ok?**



Pré modelando o problema - Estratégia sequencial

1.

Verifico a qtde total de energia ao final das integrações! a placa:

c. Stencil 5 pontos vermelha:

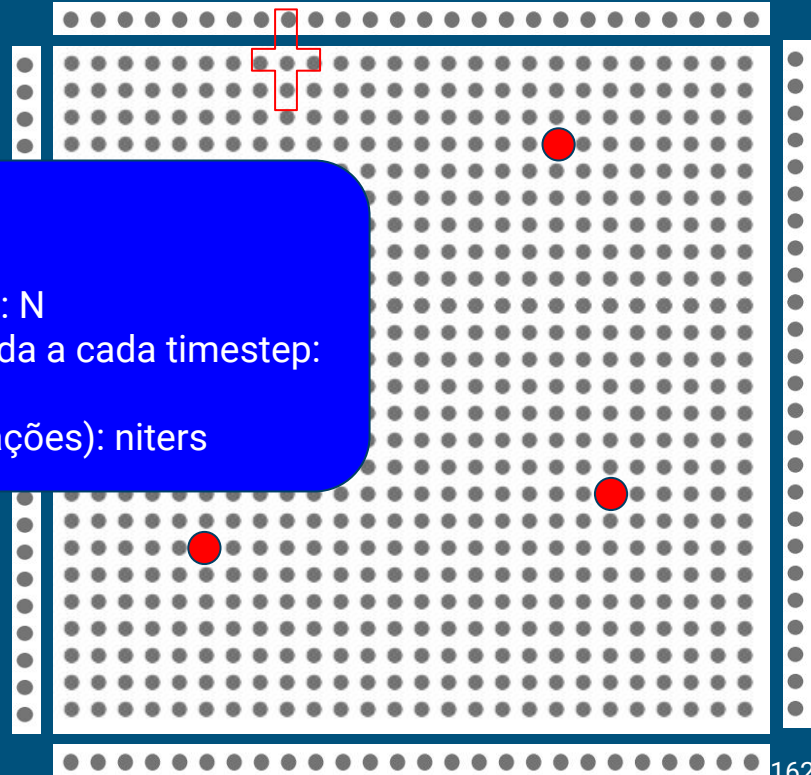
```
heat=0.0;
for(j=1; j<n+1; j++)
  for(i=1; i<n+1; i++)
    heat += aold[i][j];
```

Parâmetros:

- 1- qtde de pontos da malha: N
- 2- qtde de energia adicionada a cada timestep: energy
- 3- qtde de timesteps (interações): niters

da integração;

- f. Mas e para computar os pontos das bordas? Criar uma borda extra!
- g. Adiciona-se energia ao sistema! $+3^{\circ}\text{C}$ a cada timestep
- h. Como conferir se está tudo ok?**



Resumo da implementação sequencial

Algoritmo:

1. Cria-se as matrizes
2. Inicializa-se as fontes de energia
3. Loop principal (timestep)
 - a. $A_{new} = \text{stencil}(A_{old})$
 - b. Add energia
 - c. $A_{old} = A_{new}$
4. Imprime o total de energia

Algoritmo:

1. Cria-se as matrizes
2. Inicializa-se as fontes de energia
3. Loop principal (timestep) [2]
 - a. $A_{new} = \text{stencil}(A_{old})$
 - b. Add energia $\rightarrow A_{new}$
 - c. $A_{old} = \text{stencil}(A_{new})$
 - d. Add energia $\rightarrow A_{old}$
4. Imprime o total de energia

Algoritmo da solução:

- ~~1. Entender o mecanismos do problema;~~
- ~~2. Discretizar o problema:~~
 - Natureza é contínua
 - Computação é discreta
- ~~3. Modelar seu problema num sistema simplificado e controlado;~~
4. Executar sequencial, tempo base para cálculo de desempenho paralelo;
5. Paralelizar seu modelo;
6. Conferir resultados: versão paralela == sequencial
7. Executar versão paralela com vários cores, tomada de tempo!
8. Cálculo de desempenho

Pré modelando o problema - Estratégia sequencial

1. Modelado!
2. Implementar
3. Executar
4. Conferir resultados
5. Medir os tempos sequenciais

Qual o resultado
numérico?

Qual o tempo de
execução?

```
$ cd Mini_Curso_MPI/Ex6_Eq.Laplace/seq
```

```
Compilar: $gcc Ex6_Laplace_seq.c -o Ex6_Laplace_seq.x
```

```
Executar: $ Ex6_Laplace_seq.x [n] [energy] [niters]
```

```
$ Ex6_Laplace_seq.x 100 1 200
```

Pré modelando o problema - Estratégia sequencial

1. Modelado!
2. Implementar
3. Executar
4. Conferir resultados
5. Medir os tempos sequenciais

```
$ cd Mini_Curso_MPI/Ex6_Eq.Laplace/seq
```

```
Compilar: $gcc Ex6_Laplace_seq.c -o Ex6_Laplace_seq.x
```

```
Executar: $ Ex6_Laplace_seq.x [n] [energy] [niters]
```

```
$ Ex6_Laplace_seq.x 100 1 200
```

Qual o resultado numérico?

Qual o tempo de execução?

Varie os parâmetros de entrada, observe o comportamento do seu modelo!

Pré

cial

- 1.
- 2.
- 3.
- 4.
- 5.

- A cada timestep adiciona-se 3°C ao sistema;
- 200 iterações = 600°C no total da malha ao fim das integrações;
- Com borda (Dirichlet), ou seja, “borda fria” ($=0$), qdo o exterior está mais frio que as temperaturas na malha;
- Neste caso haverá perda de temperatura no sistema quando as temperaturas atingirem as bordas;
- Outra alternativa: as bordas como paredes isoladas, não há perda no sistema: $A[0][*] = A[2][*]$;
- CC do tipo Neumman = bordas com fluxos, definidas por alguma derivada;
- CC do tipo Robin = qdo há a combinação entre as duas CCs anteriores, deefinição direta + derivadas;

\$ cd l

Comp

Execu

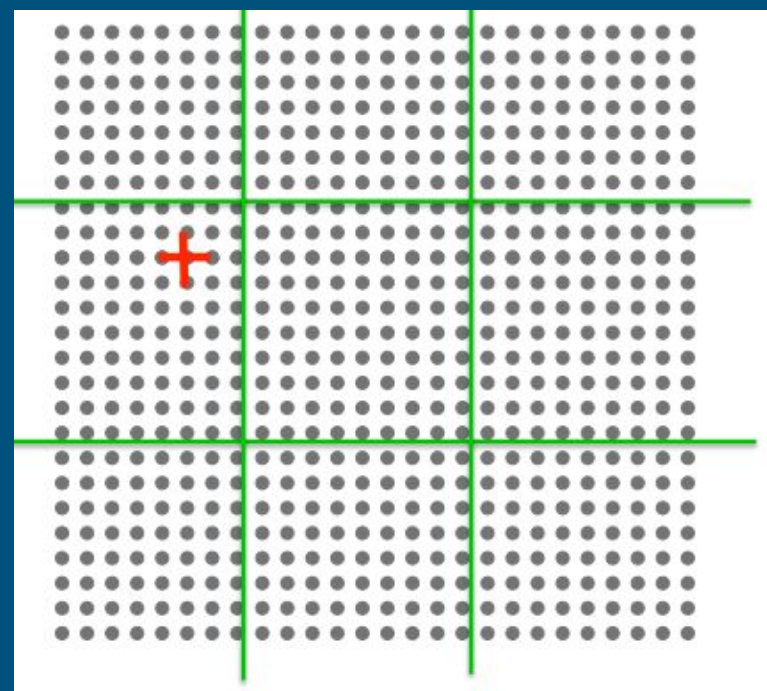
\$ Ex5_Laplace_seq.x 100 1 200

Algoritmo da solução:

- ~~1. Entender o mecanismos do problema;~~
2. Discretizar o problema:
 - a. Natureza é contínua
 - b. Computação é discreta
- ~~3. Modelar seu problema num sistema simplificado e controlado;~~
- ~~4. Executar sequencial, tempo base para cálculo de desempenho paralelo;~~
5. Paralelizar seu modelo;
6. Conferir resultados: versão paralela == sequencial
7. Executar versão paralela com vários cores, tomada de tempo!
8. Cálculo de desempenho

Estratégia paralela

- Decompor o domínio espacial em subdomínios, “chunks”...
- Cada processo recebe um pedaço para calcular
- Cada processo conhece apenas o seu subdomínio!
- Qtdes de cores = número quadrado:
 - 4 (2x2)
 - 9 (3x3)
 - 16 (4x4)
 - ...

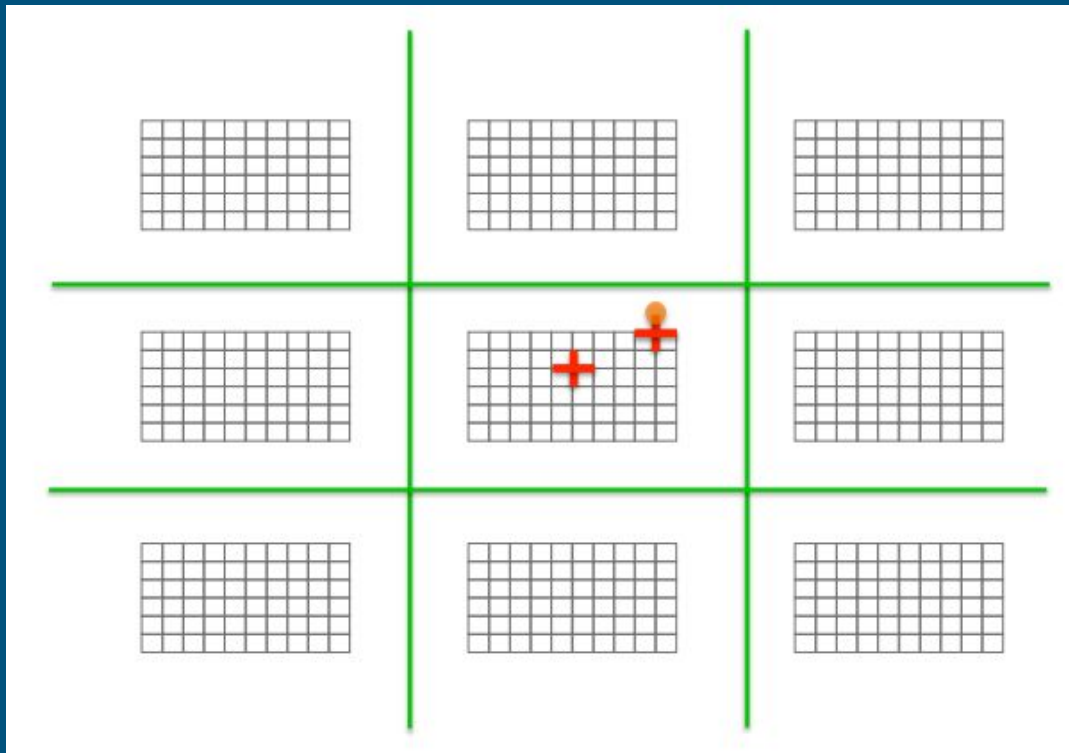


Estratégia paralela

Mesmo problema das bordas da v.seq.!

Qdo for calcular os pts nas bordas, cada core precisa conhecer o valor do ponto do seu vizinho!

NECESSIDADE DE COMUNICAÇÃO

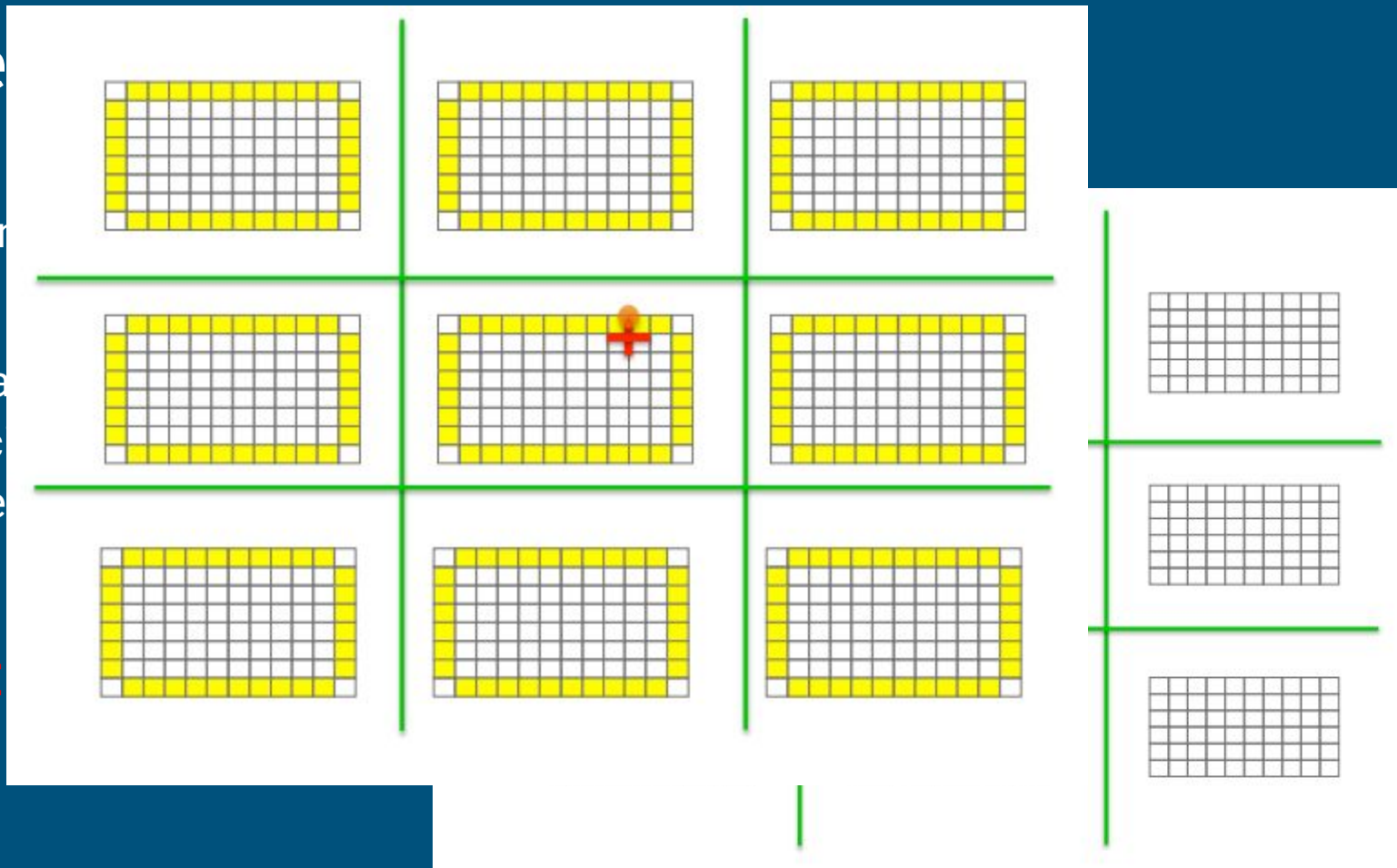


Estraté

Mesmo problema
v.seq.!

Qdo for calcular
cada core precisa
do ponto do seq

NECESSIDADE

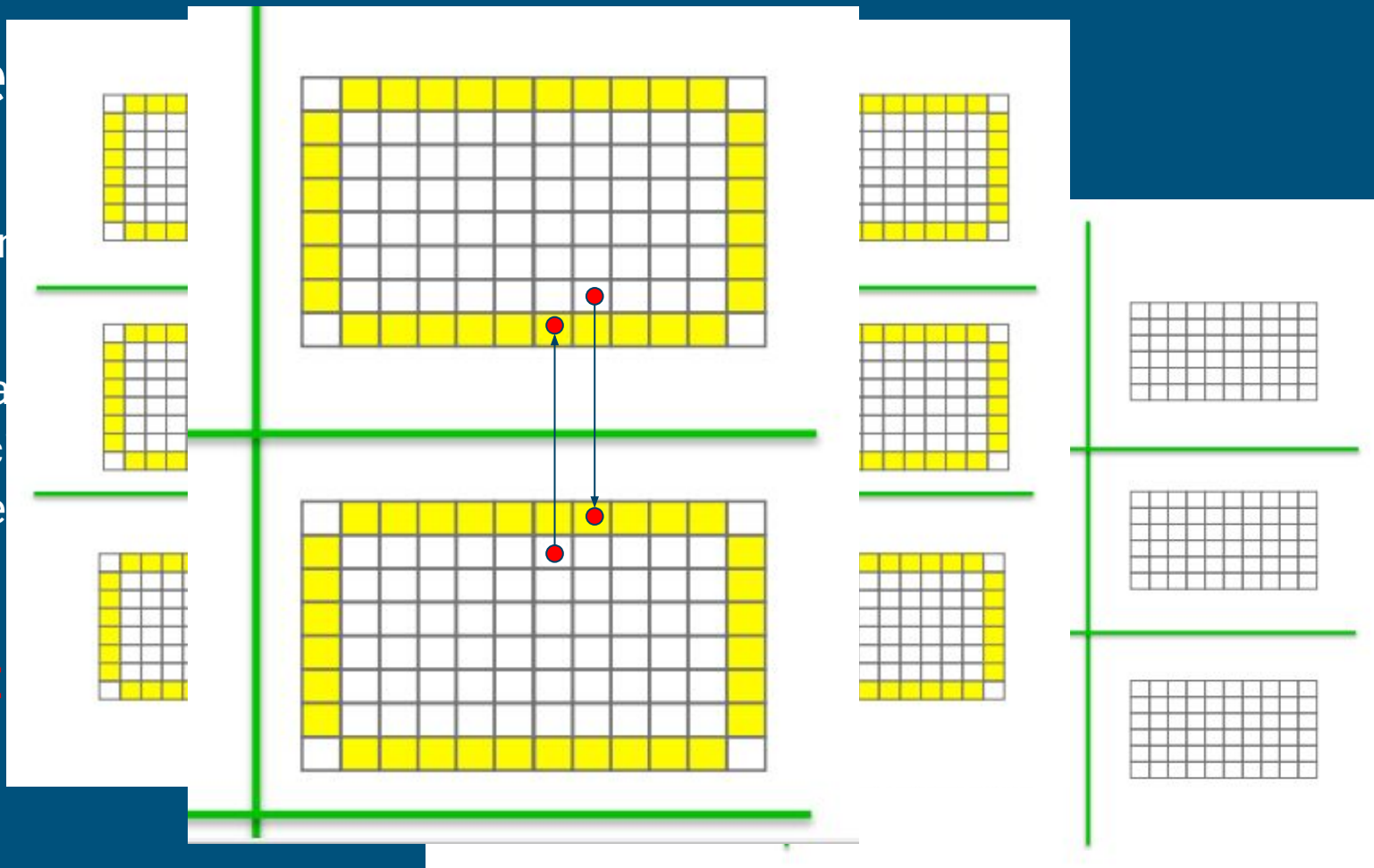


Estraté

Mesmo problema
v.seq.!

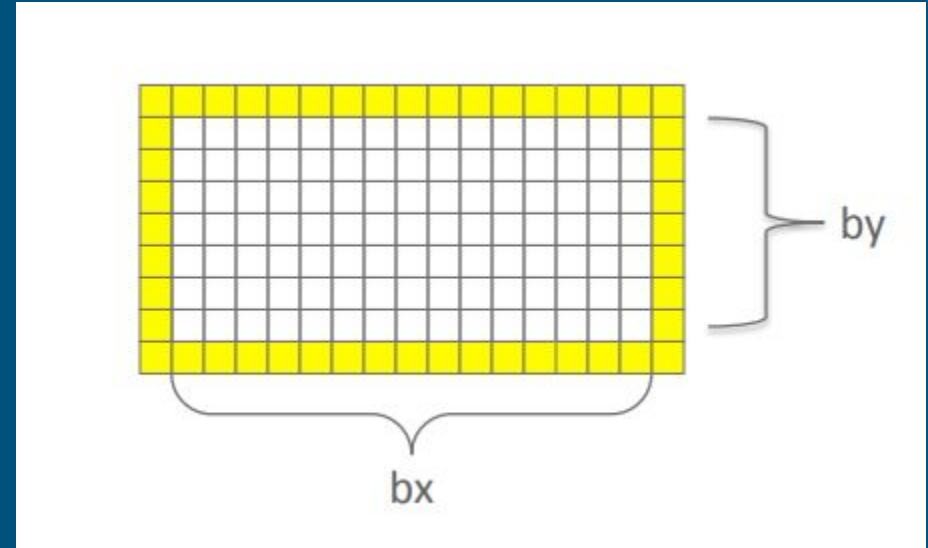
Qdo for calcular
cada core precisa
do ponto do seq

NECESSIDADE



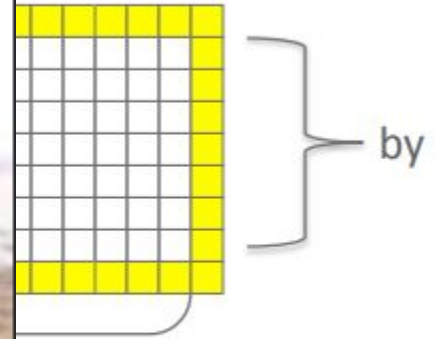
Estratégia paralela

- Cada core terá um chunk de tamanho $(bx) \times (by)$
- Considerando as ghost-zones.
- Tamanho real = $(bx+2) \times (by+2)$
- Cada core resolve o problema como na versão sequencial
- Cada core faz a somatória da qtd de energia em seu subdomínio
- Envia resultado para o “mestre”
- “Mestre” soma tudo e imprime o resultado final



Estratégia paralela

- Cada core terá um tamanho $(b_x) \times (b_y)$
- Considerando as
- Tamanho real = $(b_x) \times (b_y)$
- Cada core resolve na versão sequencial
- Cada core faz a soma da energia em seu subdomínio
- Envia resultado para o “mestre”
- “Mestre” soma tudo e imprime o resultado final



b_x

Algoritmo:

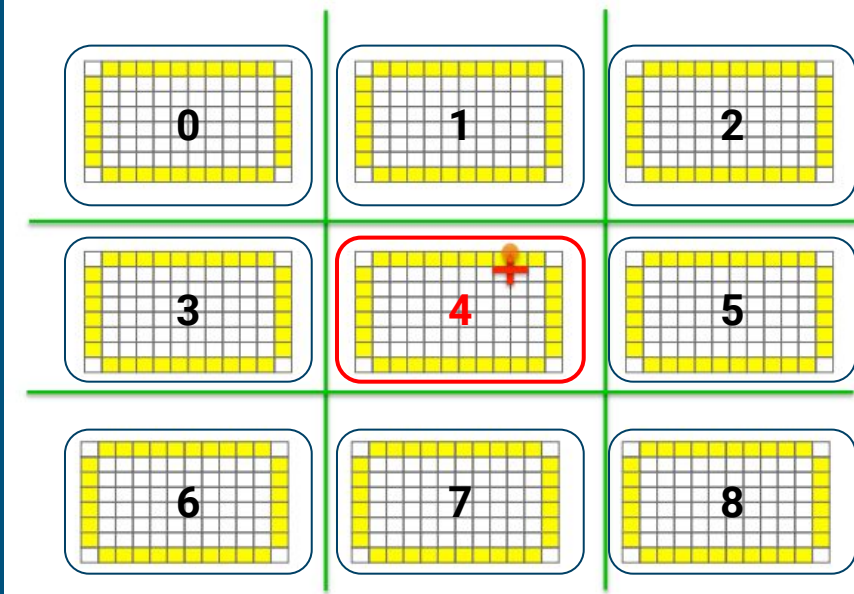
1. Divisão de domínio
2. Identifica-se os vizinhos
3. Cria-se as matrizes
4. Inicializa-se as fontes de energia
5. Loop principal (timestep)
 - a. Cria-se os pacotes (buffers) que serão usados na comunicação
 - b. Add energia
 - c. Empacota-se as bordas
 - d. Envia-se as bordas
 - e. Recebe-se as bordas
 - f. $A_{new} = \text{stencil}(A_{old})$
 - g. $A_{old} = A_{new}$
6. Mestre recebe resultados de cada core
7. Imprime o resultado

Algoritmo:

5. Loop principal (timestep)
 - a. Cria-se os pacotes (buffers)
 - b. Add energia $\rightarrow A_{old}$
 - c. Empacota-se as bordas
 - d. Envia-se as bordas
 - e. Recebe-se as bordas
 - f. $A_{new} = \text{stencil}(A_{old})$
 - g. Cria-se os pacotes (buffers)
 - h. Add energia $\rightarrow A_{new}$
 - i. Empacota-se as bordas
 - j. Envia-se as bordas
 - k. Recebe-se as bordas
 - l. $A_{old} = \text{stencil}(A_{new})$
6. Mestre recebe resultados de cada core
7. Imprime o resultado

Divisão de domínio

- Exemplo com 9 cores: 3x3 [0-8]
- Se eu for o core 4
- Quem são meus vizinhos?
 - norte :: 1
 - sul :: 7
 - leste :: 5
 - oeste :: 3
- Para saber para quem enviar minhas bordas
- De quem vou receber minhas ghost-zones

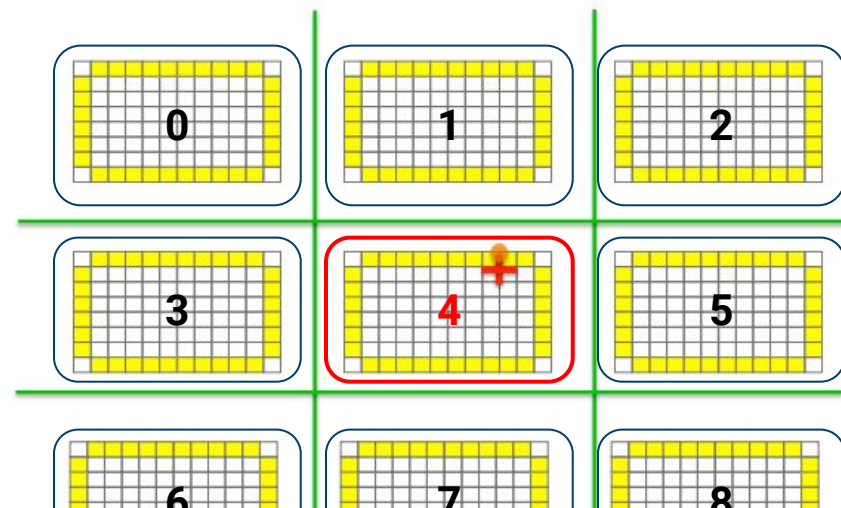


Divisão de domínio

- Exemplo com 9 cores: 3x3 [0-8]
- Se eu for o core 4
- Quem são meus vizinhos?

- norte :: 1
- sul :: 7
- leste :: 5
- oeste :: 3

- Para saber para quem e bordas
- De quem vou receber mi ghost-zones



```
// determine my coordinates (x,y) -- r=x*a+y in the 2d processor array
int rx = r % px;
int ry = r / px;
// determine my four neighbors
int north = (ry-1)*px+rx; if(ry-1 < 0) north = MPI_PROC_NULL;
int south = (ry+1)*px+rx; if(ry+1 >= py) south = MPI_PROC_NULL;
int west = ry*px+rx-1; if(rx-1 < 0) west = MPI_PROC_NULL;
int east = ry*px+rx+1; if(rx+1 >= px) east = MPI_PROC_NULL;
// decompose the domain
int bx = n/px; // block size in x
int by = n/py; // block size in y
int offx = rx*bx; // offset in x
int offy = ry*by; // offset in y
```

Divisão

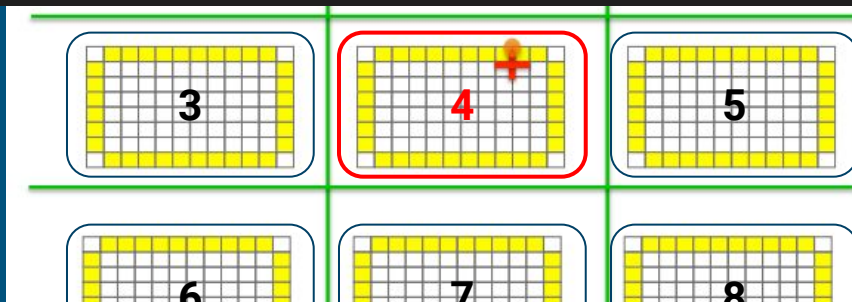
```
// allocate two work arrays
```

```
double *aold = (double*)calloc(1, (bx+2)*(by+2)*sizeof(double)); // 1-wide halo zones!
```

```
double *anew = (double*)calloc(1, (bx+2)*(by+2)*sizeof(double)); // 1-wide halo zones!
```

```
double *tmp;
```

- Exemplo com 9 cores: 3x3 [0-8]
- Se eu for o core 4
- Quem são meus vizinhos?



- norte :: 1
 - sul :: 7
 - leste :: 5
 - oeste :: 3
- Para saber para quem e
- bordas
- De quem vou receber mi
- ghost-zones

```
// determine my coordinates (x,y) -- r=x*a+y in the 2d processor array
```

```
int rx = r % px;
```

```
int ry = r / px;
```

```
// determine my four neighbors
```

```
int north = (ry-1)*px+rx; if(ry-1 < 0) north = MPI_PROC_NULL;
```

```
int south = (ry+1)*px+rx; if(ry+1 >= py) south = MPI_PROC_NULL;
```

```
int west= ry*px+rx-1; if(rx-1 < 0) west = MPI_PROC_NULL;
```

```
int east = ry*px+rx+1; if(rx+1 >= px) east = MPI_PROC_NULL;
```

```
// decompose the domain
```

```
int bx = n/px; // block size in x
```

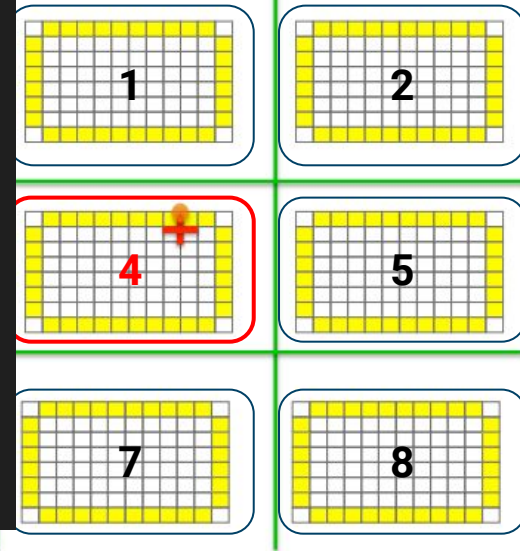
```
int by = n/py; // block size in y
```

```
int offx = rx*bx; // offset in x
```

```
int offy = ry*by; // offset in y
```



```
// initialize three heat sources
#define nsources 3
int sources[nsources][2] = {{n/2,n/2}, {n/3,n/3}, {n*4/5,n*8/9}};
int locnsources=0; // number of sources in my area
int locsources[nsources][2]; // sources local to my rank
for (int i=0; i<nsources; ++i) { // determine which sources are in my patch
    int locx = sources[i][0] - offx;
    int locy = sources[i][1] - offy;
    if(locx >= 0 && locx < bx && locy >= 0 && locy < by) {
        locsources[locnsources][0] = locx+1; // offset by halo zone
        locsources[locnsources][1] = locy+1; // offset by halo zone
        locnsources++;
    }
}
```



- sul :: 7
- leste :: 5

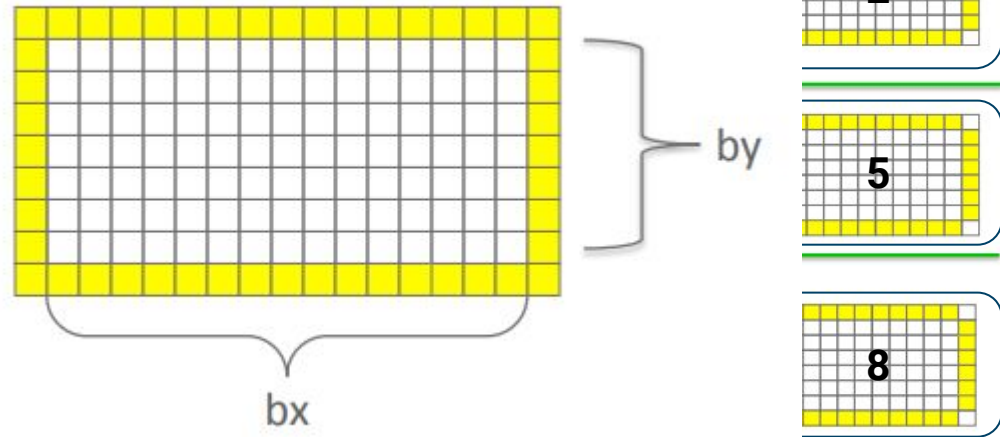
```
for(int i=0; i<locnsources; ++i) {
    aold[ind(locsources[i][0],locsources[i][1])] += energy; // heat source
}
```

bordas

- De quem vou receber minhas ghost-zones

Divisão de domínio

- Exemplo com 9 cores: 3x3 [0-8]
- Se eu for o core 4
- Quem são meus vizinhos?
 - norte :: 1
 - sul :: 7
 - leste :: 5



```
// allocate communication buffers
double *sbufnorth = (double*)calloc(1,bx*sizeof(double)); // send buffers
double *sbufsouth = (double*)calloc(1,bx*sizeof(double));
double *sbufeast = (double*)calloc(1,by*sizeof(double));
double *sbufwest = (double*)calloc(1,by*sizeof(double));
double *rbufnorth = (double*)calloc(1,bx*sizeof(double)); // receive buffers
double *rbufsouth = (double*)calloc(1,bx*sizeof(double));
double *rbufeast = (double*)calloc(1,by*sizeof(double));
double *rbufwest = (double*)calloc(1,by*sizeof(double));
```



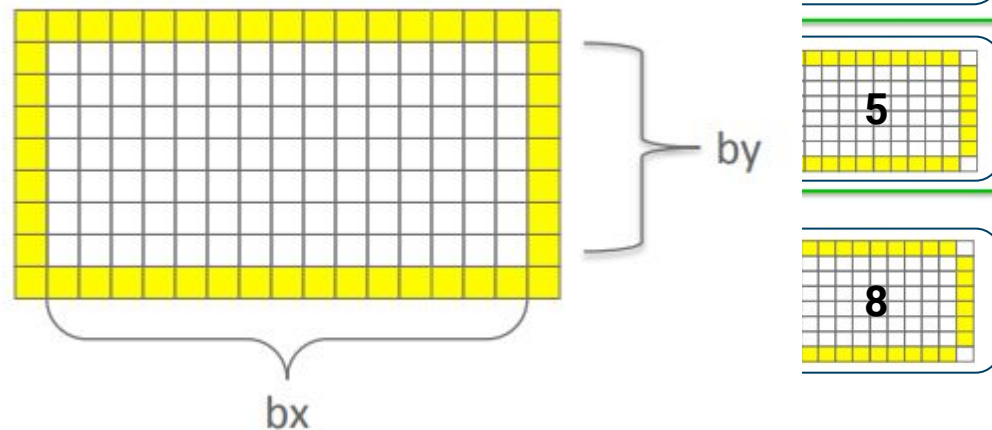
```

for(int i=0; i<bx; ++i) sbufnorth[i] = aold[ind(i+1,1)]; // pack loop - last valid regi
for(int i=0; i<bx; ++i) sbufsouth[i] = aold[ind(i+1,by)]; // pack loop
for(int i=0; i<by; ++i) sbufeast[i] = aold[ind(bx,i+1)]; // pack loop
for(int i=0; i<by; ++i) sbufwest[i] = aold[ind(1,i+1)]; // pack loop

```

Divisao de dominio

- Exemplo com 9 cores: 3x3 [0-8]
- Se eu for o core 4
- Quem são meus vizinhos?
 - norte :: 1
 - sul :: 7
 - leste :: 5



```

// allocate communication buffers
double *sbufnorth = (double*)calloc(1,bx*sizeof(double)); // send buffers
double *sbufsouth = (double*)calloc(1,bx*sizeof(double));
double *sbufeast = (double*)calloc(1,by*sizeof(double));
double *sbufwest = (double*)calloc(1,by*sizeof(double));
double *rbufnorth = (double*)calloc(1,bx*sizeof(double)); // receive buffers
double *rbufsouth = (double*)calloc(1,bx*sizeof(double));
double *rbufeast = (double*)calloc(1,by*sizeof(double));
double *rbufwest = (double*)calloc(1,by*sizeof(double));

```

```

for(int i=0; i<bx; ++i) sbufnorth[i] = aold[ind(i+1,1)]; // pack loop - last valid regi
for(int i=0; i<bx; ++i) sbufsouth[i] = aold[ind(i+1,by)]; // pack loop
for(int i=0; i<by; ++i) sbufeast[i] = aold[ind(bx,i+1)]; // pack loop
for(int i=0; i<by; ++i) sbufwest[i] = aold[ind(1,i+1)]; // pack loop

```

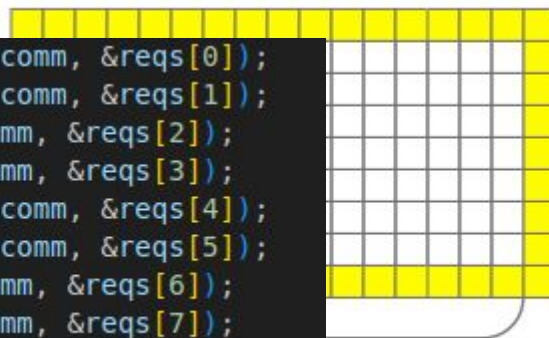
Divisao de domini

```

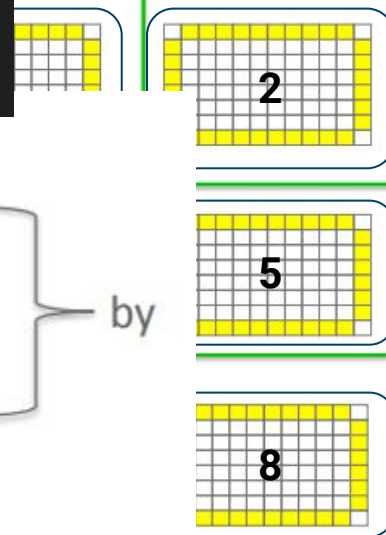
MPI_Isend(sbufnorth, bx, MPI_DOUBLE, north, 9, comm, &reqs[0]);
MPI_Isend(sbufsouth, bx, MPI_DOUBLE, south, 9, comm, &reqs[1]);
MPI_Isend(sbufeast, by, MPI_DOUBLE, east, 9, comm, &reqs[2]);
MPI_Isend(sbufwest, by, MPI_DOUBLE, west, 9, comm, &reqs[3]);
MPI_Irecv(rbufnorth, bx, MPI_DOUBLE, north, 9, comm, &reqs[4]);
MPI_Irecv(rbufsouth, bx, MPI_DOUBLE, south, 9, comm, &reqs[5]);
MPI_Irecv(rbufeast, by, MPI_DOUBLE, east, 9, comm, &reqs[6]);
MPI_Irecv(rbufwest, by, MPI_DOUBLE, west, 9, comm, &reqs[7]);
MPI_Waitall(8, reqs, Allstats);

```

- sul :: 7
- leste :: 5



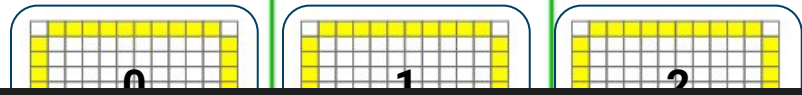
bx



```

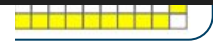
// allocate communication buffers
double *sbufnorth = (double*)calloc(1,bx*sizeof(double)); // send buffers
double *sbufsouth = (double*)calloc(1,bx*sizeof(double));
double *sbufeast = (double*)calloc(1,by*sizeof(double));
double *sbufwest = (double*)calloc(1,by*sizeof(double));
double *rbufnorth = (double*)calloc(1,bx*sizeof(double)); // receive buffers
double *rbufsouth = (double*)calloc(1,bx*sizeof(double));
double *rbufeast = (double*)calloc(1,by*sizeof(double));
double *rbufwest = (double*)calloc(1,by*sizeof(double));

```



```
// update grid points
heat = 0.0;
for(int j=1; j<by+1; ++j) {
  for(int i=1; i<bx+1; ++i) {
    anew[ind(i,j)] = aold[ind(i,j)]/2.0 + (aold[ind(i-1,j)] + aold[ind(i+1,j)] + aold[ind(i,j-1)] + aold[ind(i,j+1)])/4.0/2.0;
    heat += anew[ind(i,j)];
  }
}

// swap arrays
tmp=anew; anew=aold; aold=tmp;
```

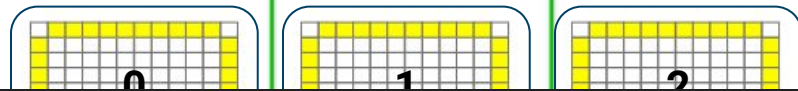


- norte :: 1
- sul :: 7
- leste :: 5
- oeste :: 3

bx

- Para saber as bordas
- De quem são as ghost-zones

```
// get final heat in the system
double rheat;
MPI_Allreduce(&heat, &rheat, 1, MPI_DOUBLE, MPI_SUM, comm);
if(!r) printf("[%i] last heat: %f time: %f\n", r, rheat, t);
```



```
// update grid points
heat = 0.0;
for(int j=1; j<by+1; ++j) {
  for(int i=1; i<bx+1; ++i) {
    anew[ind(i,j)] = aold[ind(i,j)]/2.0 + (aold[ind(i-1,j)] + aold[ind(i+1,j)] + aold[ind(i,j-1)] + aold[ind(i,j+1)])/4.0/2.0;
    heat += anew[ind(i,j)];
  }
}

// swap arrays
tmp=anew; anew=aold; aold=tmp;
```

Pronto! Agora é só rodar!



- Para saber as bordas
- De quem são as ghost-zones

```
// get final heat in the system
double rheat;
MPI_Allreduce(&heat, &rheat, 1, MPI_DOUBLE, MPI_SUM, comm);
if(!r) printf("[%i] last heat: %f time: %f\n", r, rheat, t);
```

Executando versão paralela

1. Executar
2. Conferir resultados
3. Medir os tempos paralelos

```
$ cd Mini_Curso_MPI/Ex6_Eq.Laplace/mpi/v0
```

```
Compilar: $ mpicc Ex6_Laplace_mpi_ISIRv0.c -o Ex6_Laplace_mpi_ISIRv0.x
```

```
Executar: $ mpirun -n 4 Ex6_Laplace_mpi_ISIRv0.x [n] [energy] [niters] [npx] [npy]
```

```
$ mpirun -n 4 Ex6_Laplace_mpi_ISIRv0.x 100 1 200 2 2
```


Executando versão paralela

1. Executar
2. Conferir resultados
3. Medir os tempos paralelos

Qual o resultado numérico?

Varie os parâmetros de entrada, observe o comportamento do seu modelo!

Qual o tempo de execução?

```
$ cd Mini_Curso_MPI/Ex6_Eq.Laplace/mpi/v0
```

```
Compilar: $ mpicc Ex6_Laplace_mpi_ISIRv0.c -o Ex6_Laplace_mpi_ISIRv0.x
```

```
Executar: $ mpirun -n 4 Ex6_Laplace_mpi_ISIRv0.x [n] [energy] [niters] [npx] [npy]
```

```
$ mpirun -n 4 Ex6_Laplace_mpi_ISIRv0.x 100 1 200 2 2
```

Mod1: versão paralela

1. Explodir o código! (??)
 - a. Temporizar trechos do código!
 - b. Usando `MPI_Wtime()`;

- Vamos medir os trechos:
 - do cálculo do stencil
 - da comunicação
 - atualização das matrizes

`MPI_WTIME` returns a floating-point number of seconds, representing elapsed wall-clock time since some time in the past.

Mod1: versão paralela - executando

```
$ cd Mini_Curso_MPI/Ex6_Eq.Laplace/mpi/v1
```

```
Compilar: $ mpicc Ex6_Laplace_mpi_ISIRv1.c -o Ex6_Laplace_mpi_ISIRv1.x
```

```
Executar: $ mpirun -n 4 Ex6_Laplace_mpi_ISIRv1.x [n] [energy] [niters] [npx] [npy]
```

```
$ mpirun -n 4 Ex6_Laplace_mpi_ISIRv1.x 100 1 200 2 2
```


Mod1: versão paralela - executando

```
$ cd Mini_Curso_MPI/Ex6_Eq.Laplace/mpi/v1
```

```
Compilar: $ mpicc Ex6_Laplace_mpi_ISIRv1.x
```

Como ficou o
desempenho do nosso
modelo paralelo?

```
Executar: $ mpirun -n 4 Ex6_Laplace_mpi_ISIRv1.x [n] [energy] [niters] [npx] [npy]
```

```
$ mpirun -n 4 Ex6_Laplace_mpi_ISIRv1.x 100 1 200 2 2
```

Mod1: vers

lo

Calcular o Speedup e Eficiência!

Plotar os gráficos!

\$ cd Mini_Curso_MPI/

Compilar: \$ mpicc Ex6_Laplace.

_ISIRv1.x

Como ficou o
desempenho do nosso
modelo paralelo?

Executar: \$ mpirun -n 4 Ex6_Laplace_mpi_ISIRv1.x [n] [energy] [niters] [npx] [npv]

\$ mpirun -n 4 Ex6_Laplace_mpi_ISIRv1.x 100 1 200 2 2