

# Programação Paralela: das *threads* aos FPGAs

## Conceitos e Terminologia

Prof. Ricardo Menotti  
[menotti@ufscar.br](mailto:menotti@ufscar.br)

Prof. Maurício Acconcia Dias  
[macccdias@gmail.com](mailto:macccdias@gmail.com)

Prof. Helio Crestana Guardia  
[helio.guardia@ufscar.br](mailto:helio.guardia@ufscar.br)

Departamento de Computação  
Universidade Federal de São Carlos

Atualizado em: 10 de maio de 2020

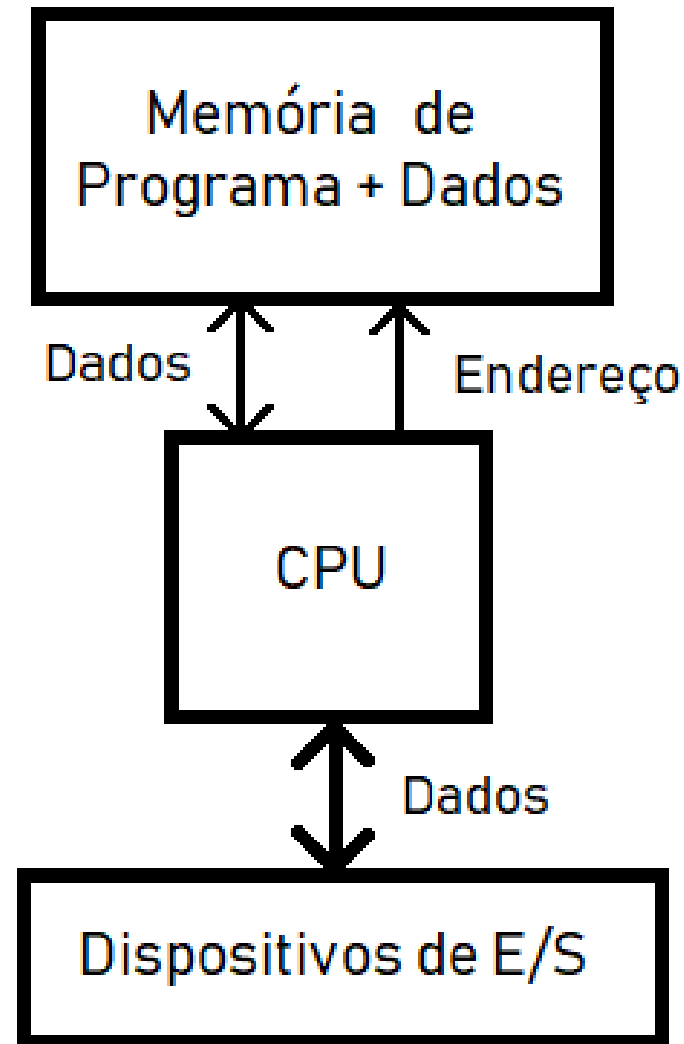


# Agenda

- Arquitetura de computadores
  - Von Neumann x Harvard
- Taxonomia de Flynn
- Conceitos e terminologia
- Limites e custos da programação paralela
- Arquiteturas de Memória
  - Compartilhada
  - Distribuída
  - Híbrida

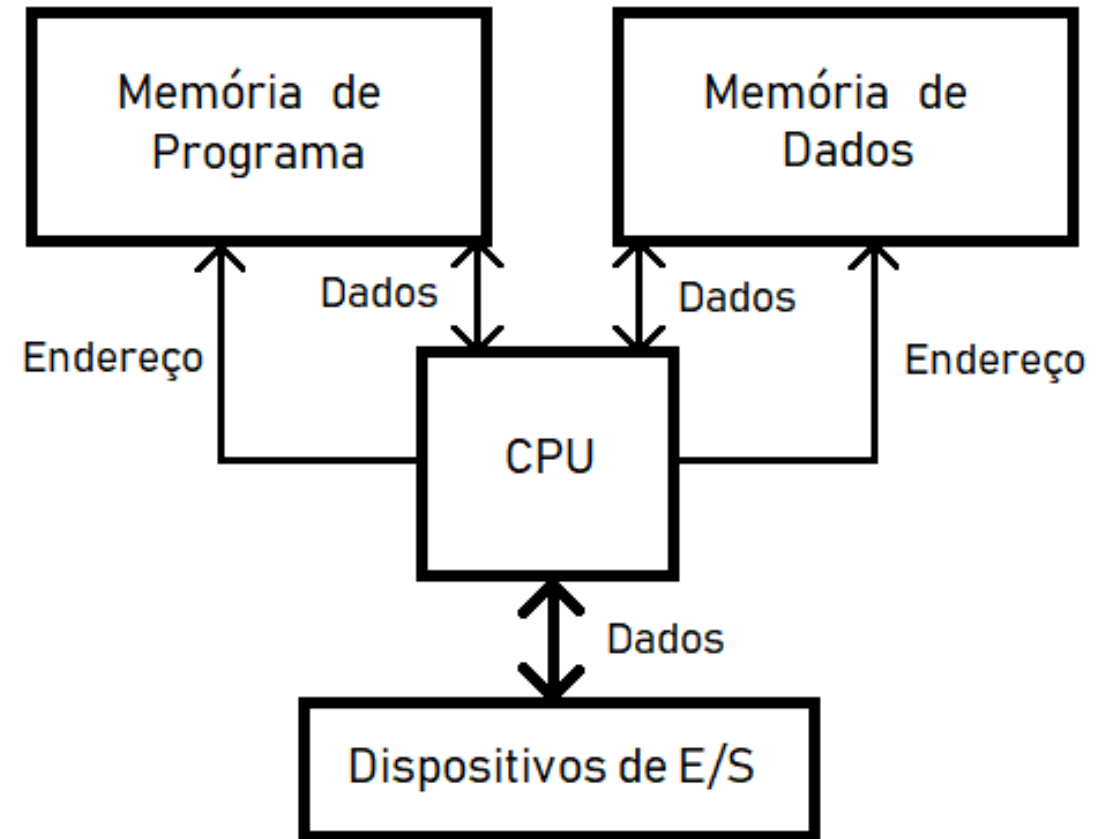
# Arquitetura de von Neumann

- Apenas uma memória
- Arquitetura mais simples
- Tempo de execução maior
- Hardware final com custo mais baixo
- Geralmente CISC



# Arquitetura de harvard

- Memórias separadas
- Arquitetura mais complexa
- Tempo de execução menor
- Hardware final com custo mais Elevado
- Geralmente RISC



# Questões Importantes para o Paralelismo

- A arquitetura da maioria dos computadores programados para a execução de algoritmos paralelos é baseada na arquitetura de von Neumann
- A unidade de controle decodifica as operações e às executa de forma sequencial
- Replicar este comportamento para obter a execução paralela de instruções resulta em problemas e essa é uma questão a se pensar.

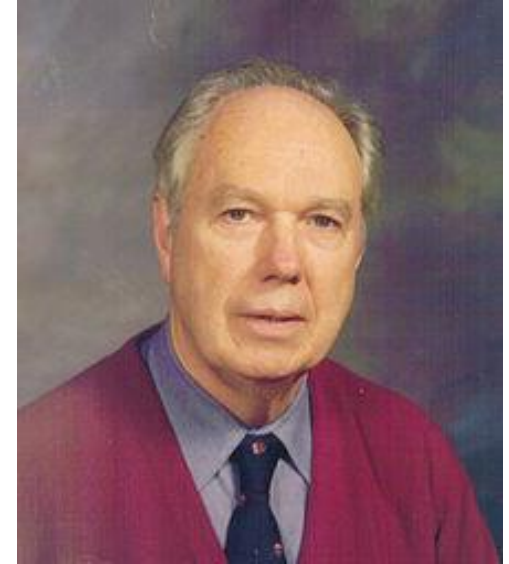
# Taxonomia

# Classificação de computadores paralelos

- Existem diversas maneiras de se organizar uma arquitetura para a execução paralela
- De acordo com o estudo a ser realizado e o resultado que se deseja obter uma maneira pode ser mais produtiva que outra para a classificação
- Independente disso, umas das classificações mais utilizadas é a taxonomia de Flynn

# Taxonomia de Flynn

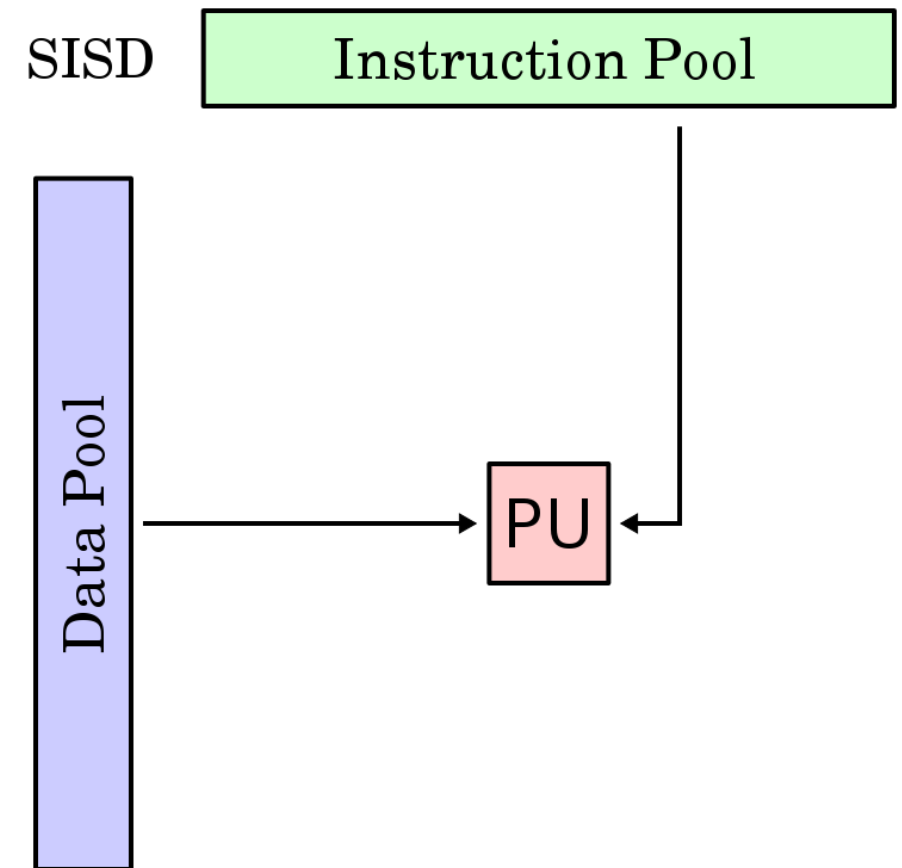
- Prof. Michael J Flynn
  - Professor emérito da universidade de Stanford
- Criou uma das taxonomias mais utilizadas na área de computação para máquinas paralelas
- Como classificava processadores, sua taxonomia é baseada em número de instruções e fluxo de dados, possuindo 4 categorias





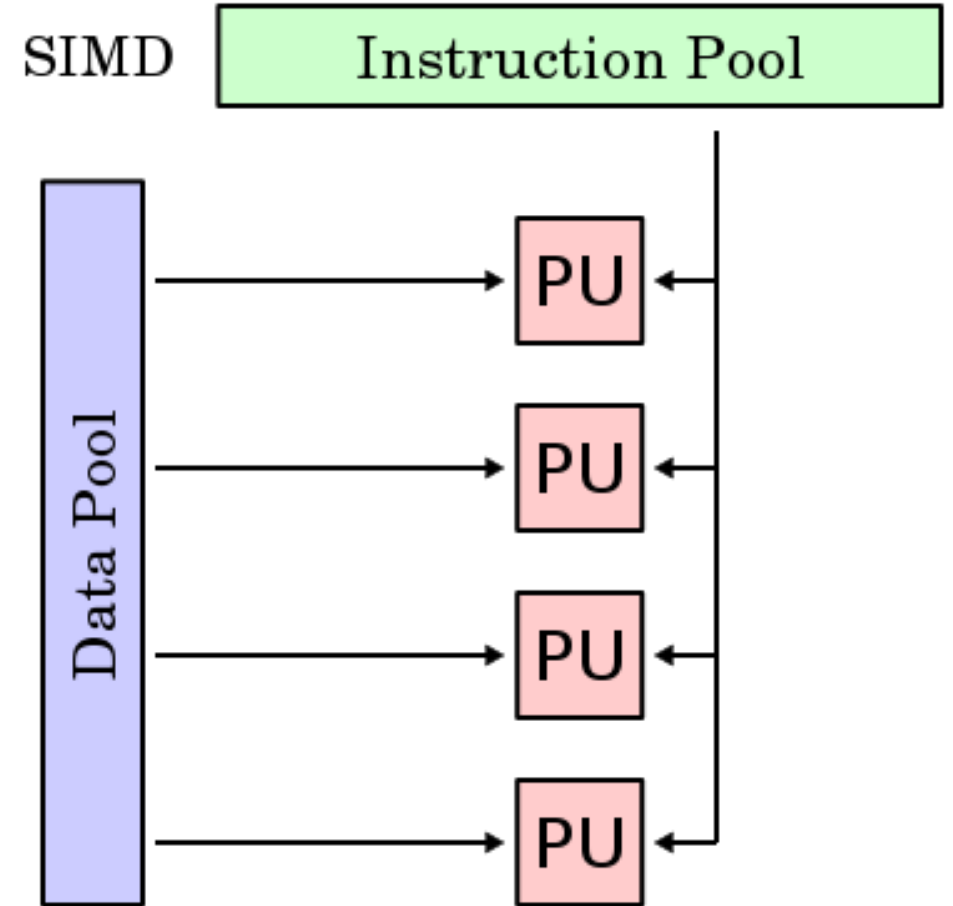
# Taxonomia de Flynn

- SISD – *Single Instruction Single Data*
- Apenas uma instrução é executada sobre um conjunto de dados determinado
- Arquitetura presente em computadores *single-core*



# Taxonomia de Flynn

- SIMD – *Single Instruction Multiple Data*
- Uma única instrução opera sobre um grande conjunto de dados
- Exemplo: operação executada em todos os elementos de um vetor como o ajuste de contraste em imagens ou de volume em áudio



# Pausa para o SMT e SIMT

- *Threads* são unidades básicas de processamento originadas de um processo. Compartilham a memória do processo “pai” e costumam executar pequenos trechos de código
- O que fazer quando existem duas ULAs disponíveis no processador (e.g., Pentium 4)?
  - Executar uma *thread* em cada ULA dando a impressão de dois núcleos
  - Tecnologia chamada de *Hyper-Threading* pela intel
  - Classificada como SMT – *Simultaneous multi-threading*

# Pausa para o SMT e SIMT

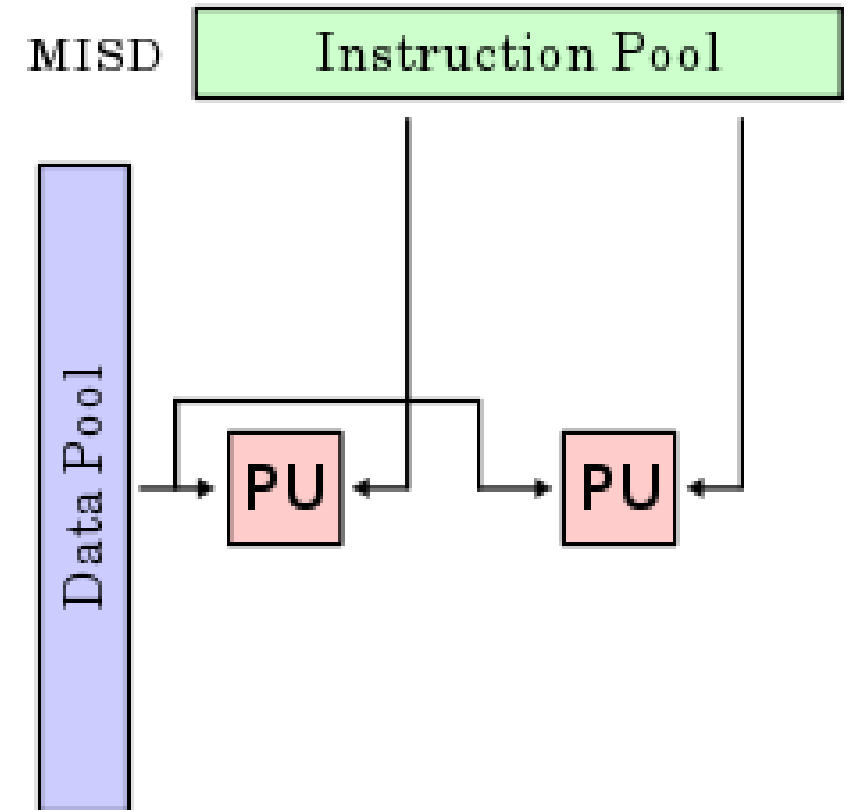
- E quando a *thread* fica bloqueada?
  - Meio processador para de funcionar (chamaremos de pseudo-núcleo)
- Esta tecnologia é gerenciada por uma unidade especial
  - Chamada OoO – *Out of Order* unit
- A solução encontrada foi o que é definido como SIMT – *Single Instruction Multiple Thread*
  - Na prática, quando há um bloqueio a unidade OoO para de executar

# Pausa para o SMT e SIMT



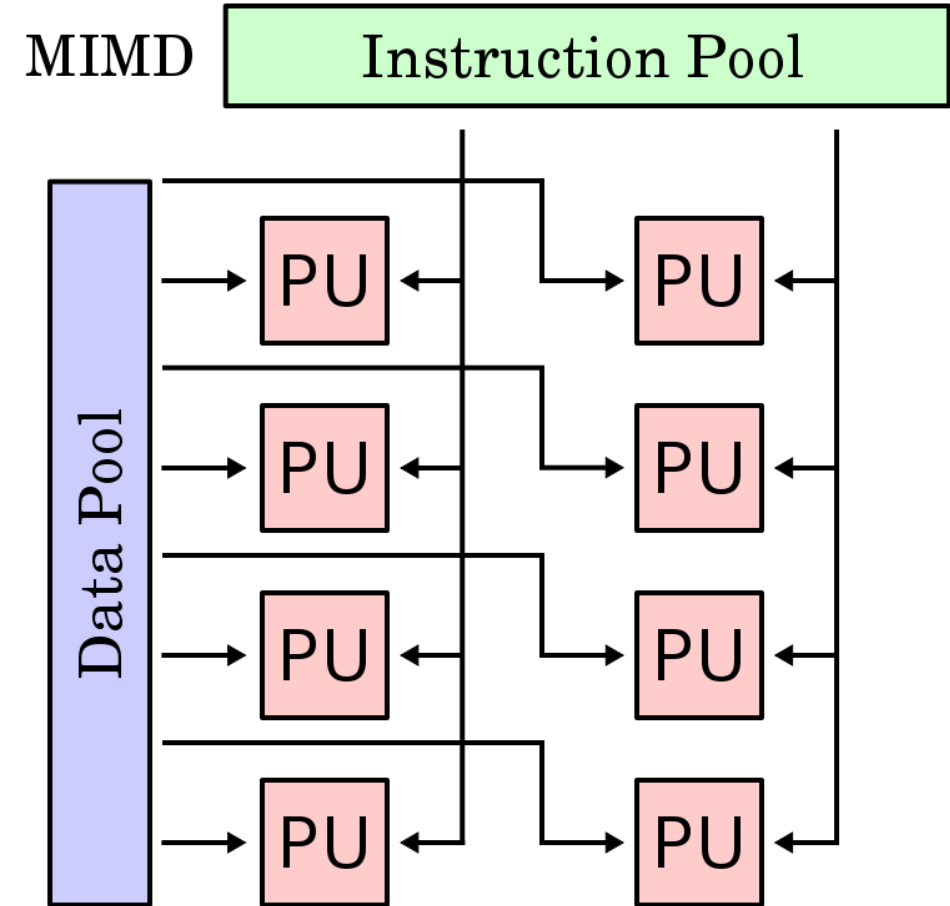
# Taxonomia de Flynn

- MISD – *Multiple Instruction Single Data*
- Este é um modelo que não encontra ampla aplicação prática comparado com os outros tipos
- Porém, é o modelo que pode ser aplicado a casos de tolerância a falhas e criptografia



# Taxonomia de Flynn

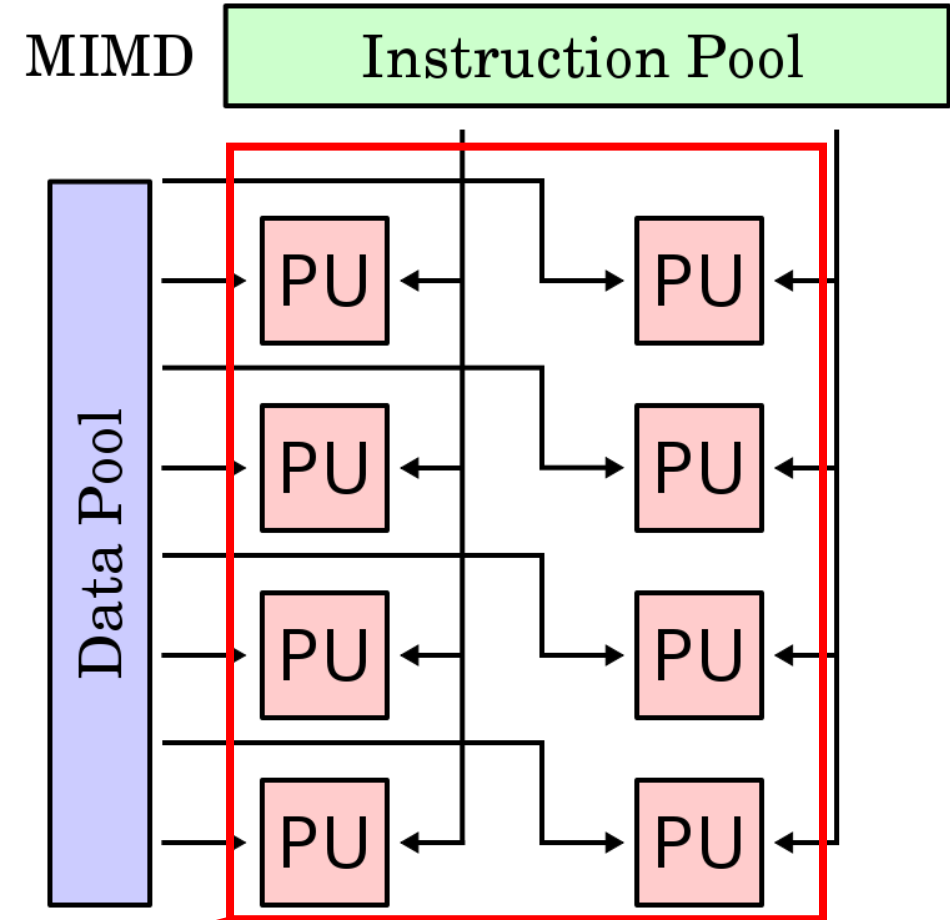
- MIMD – *Multiple Instruction Multiple Data*
- Modelo da “moda” =)
- Multicores
- Processadores superescalares
- Sistemas distribuídos



# Taxonomia de Flynn

- MIMD – *Multiple Instruction Multiple Data*
- Modelo da “moda” =)
- Multicores
- Processadores superescalares
- Sistemas distribuídos

Múltiplas Unidades de  
Processamento





# Terminologia

# Terminologia

- Supercomputador/HPC – High Performance Computer
  - Computadores grandes e rápidos para solucionar problemas
- Nó
  - Um computador sozinho. Um conjunto de nós interligados, por exemplo em rede, formam um supercomputador
- CPU/Socket/Processador/Núcleo (Core)
  - Unidade básica de processamento do modelo em questão
- Tarefa
  - Unidade básica de trabalho a ser executado por um nó

# Terminologia

- Pipelining
  - Quebra de uma tarefa para poder ser executada de forma modular
- Memória compartilhada
  - Todas as unidades de processamento possuem acesso a toda a memória do sistema
- Memória Distribuída
  - Cada unidade de processamento tem sua memória que só pode ser acessada por comunicação
- Comunicação
  - Forma utilizada pelo nós para troca de informações

# Terminologia

- Sincronização
  - Coordenação de execução de tarefas paralelas em tempo real
- Granularidade
  - Em programação paralela é a taxa de comunicação entre nós
    - Fina - pequenas trocas de dados
    - Grossa – grande quantidade de informação é trocada
- Speedup
  - Tempo da execução sequencial/tempo da execução paralela
  - Indicador simples de desempenho (porém um pouco problemático)

# Terminologia

- *Overhead* paralelo
  - Tempo gasto para a gerência da execução paralela de tarefas
    - Controle de tempo
    - Sincronização
    - Comunicação
    - Características particulares de bibliotecas de execução paralela
- Massivamente paralelo
  - Quando um determinado hardware possui muitas unidades de processamento e consegue executar uma tarefa de forma extremamente paralela

# Terminologia

- Escalabilidade
  - Capacidade de um sistema paralelo de melhorar o speedup da aplicação com o aumento do número de unidades de processamento envolvidas na computação
- Esta característica não é simples de ser atingida e depende de
  - Hardware
  - Algoritmo
  - Overhead paralelo
  - Características da aplicação

Limitações

# Limitações

- Nem tudo são flores na computação paralela....
- Principalmente na hora de comparar ou estimar resultados
- O *Speedup* pela definição básica é uma medida rápida, porém falha de medir a melhora proporcionada por um algoritmo paralelo
  - Não considera absolutamente nada a não ser o tempo de execução

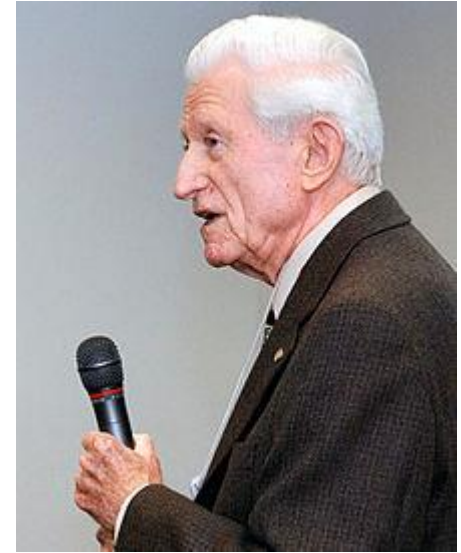


# Limitações

- Por isso usamos a lei de Amdahl
- Vamos então à equação completa

$$speedup = \frac{1}{\frac{P}{N} + S}$$

- P é a % do código em paralelo
- N é o número de processadores
- S é a % do seu código que continua sequencial

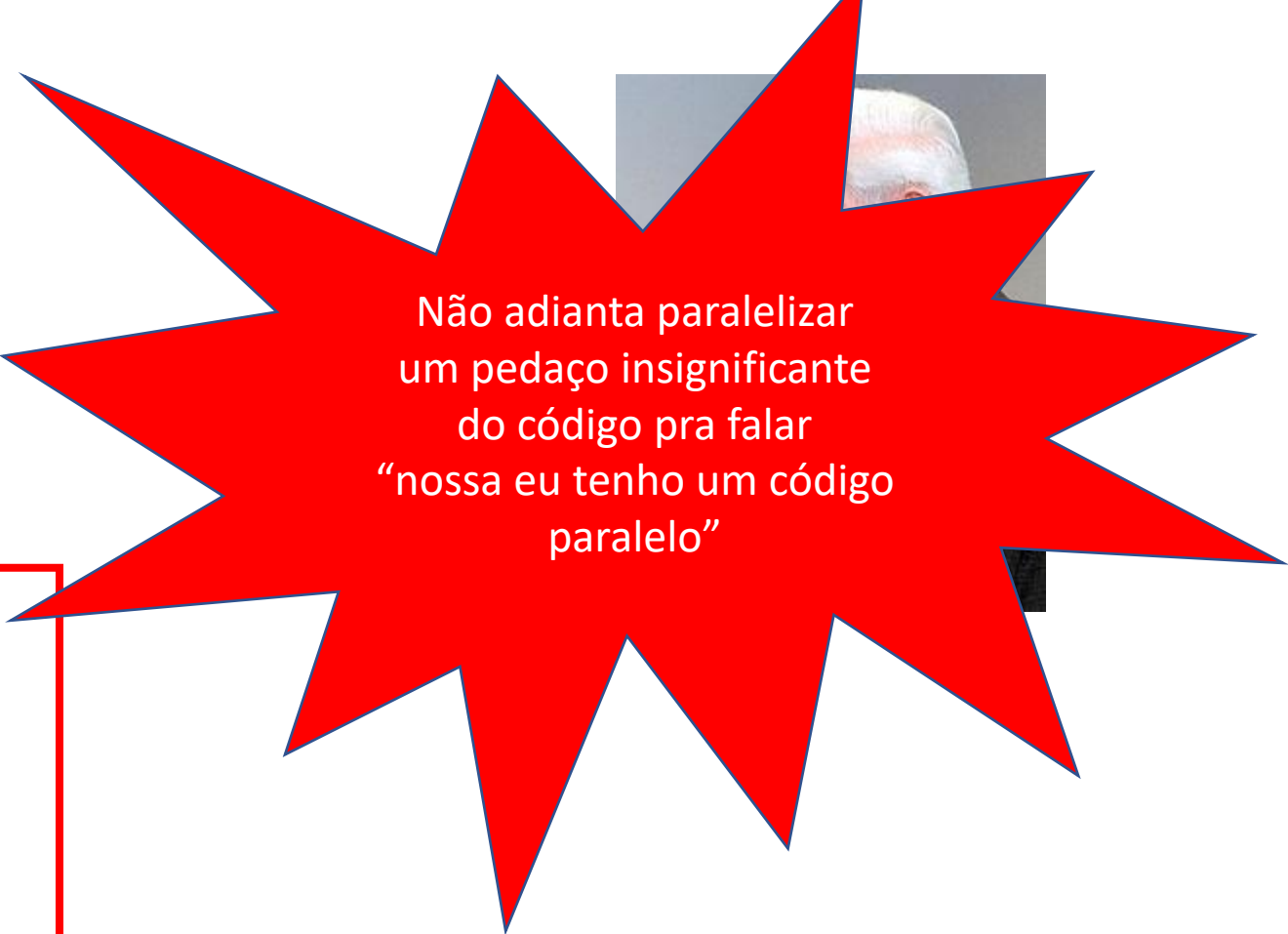


# Limitações

- Por isso usamos a lei de Amdahl
- Vamos então à equação completa

$$speedup = \frac{1}{\frac{P}{N} + S}$$

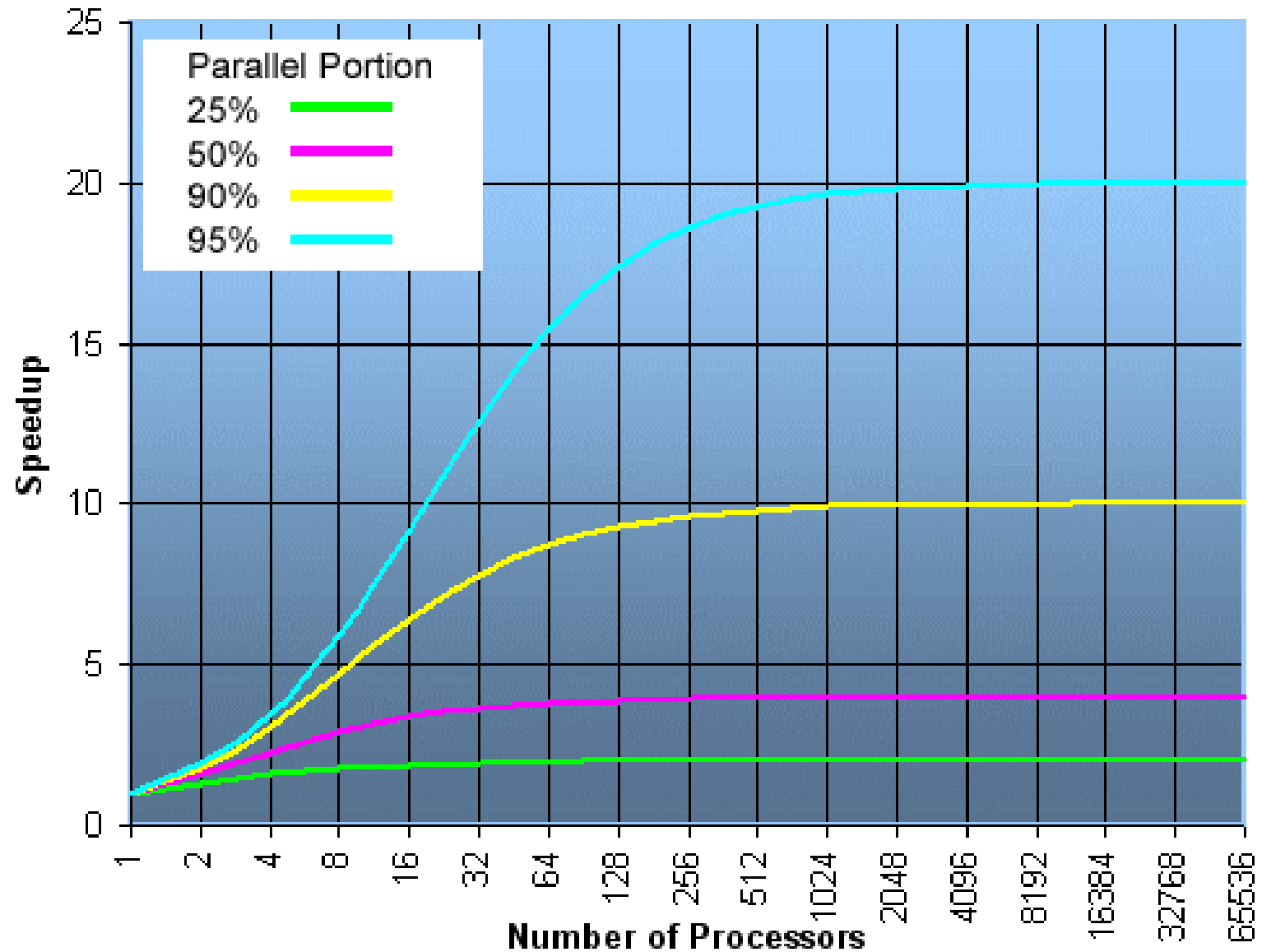
- P é a % do código em paralelo
- N é o número de processadores
- S é a % do seu código que continua sequencial



Não adianta paralelizar  
um pedaço insignificante  
do código pra falar  
“nossa eu tenho um código  
paralelo”

# Limitações

- Veja o sugestivo comportamento da curva



# Limitações

- Veja o sugestivo comportamento da curva



# Limitações

- O segundo problema é a complexidade
- Desenvolver um algoritmo paralelo é substancialmente mais complexo, aumentando principalmente os custos de:
  - Projeto
  - Codificação
  - Debug
  - Teste
  - Manutenção

# Limitações

- O terceiro é a portabilidade
  - Códigos paralelos são portáveis igual Java
- Apesar de uma melhora considerável nas últimos anos...
  - As bibliotecas podem apresentar “melhorias” que irão impedir sua execução fora do ambiente original
  - Apesar dos padrões, as aplicações exigem em alguns casos soluções específicas
  - Sistemas operacionais
  - Hardware – requisitos de memória, processamento, overhead paralelo



# Limitações

- Escalabilidade:
- Forte – problema fixo, processadores são adicionados
  - Se for bem implementado o algoritmo, com mais processadores o tempo de execução irá cair em alguma proporção.
- Fraco – tamanho do problema por processador é fixo
  - Ou seja, o tamanho total é proporcional ao número de processadores utilizados
  - Objetivo é rodar um problema mais complexo no mesmo tempo

# Limitações

- Ingenuamente pensamos que adicionais mais processadores irá fazer com que o tempo de processamento caia
  - Em alguns casos além de não cair, piora
  - Aumentar o número de processadores raramente será a resposta
- Motivos
  - Tempo de comunicação cpu-memória
  - Overhead de comunicação
  - Pouca memória disponível
  - Clock do processador



Memória

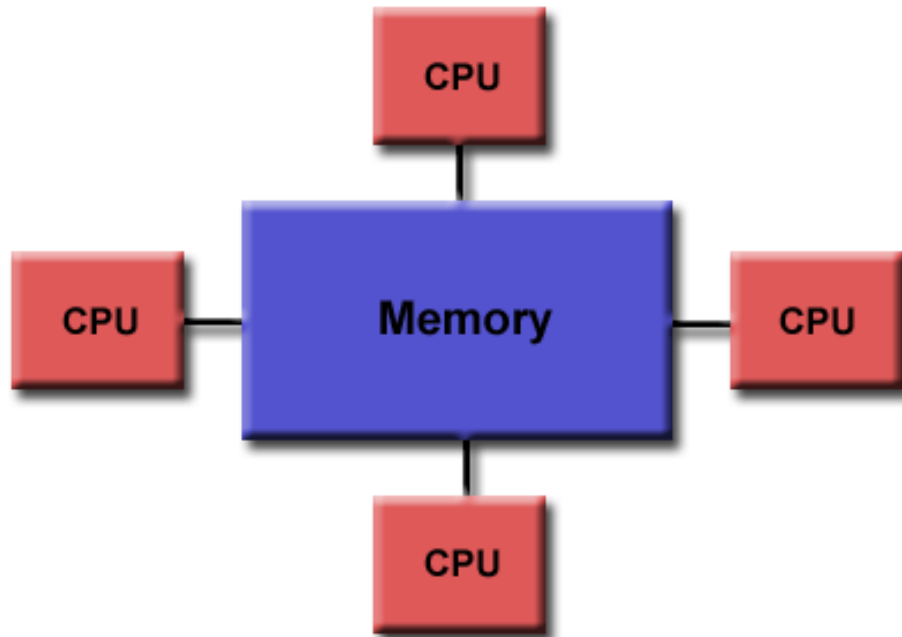
# Arquiteturas de memória

- Em sistemas paralelos podemos implementar 3 tipos de memória
  - Compartilhada
  - Distribuída
  - Híbrida

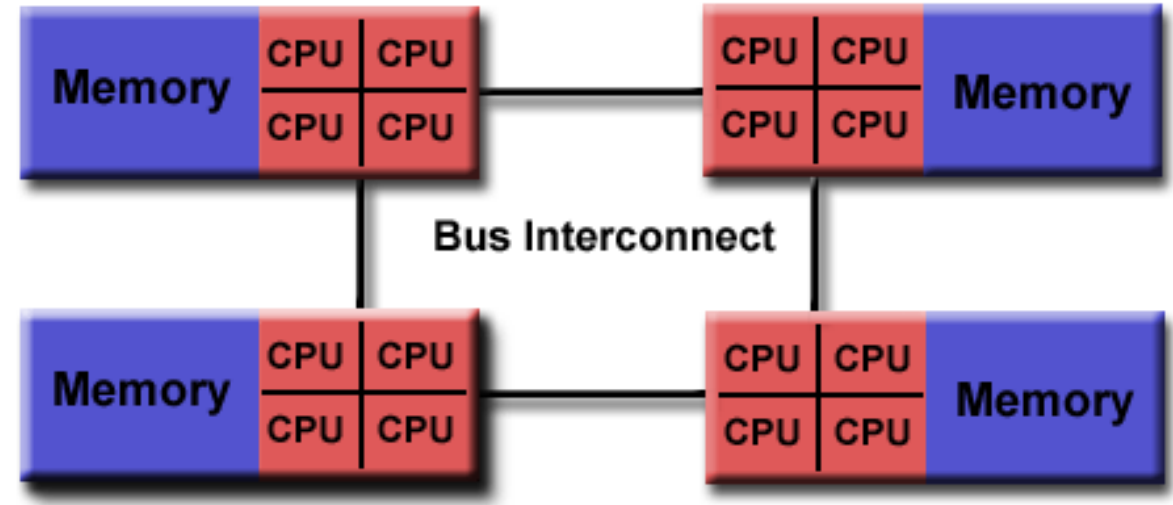
# Arquiteturas de memória

- Memória compartilhada

UMA – Uniform Memory Access



NUMA – Non-Uniform Memory Access

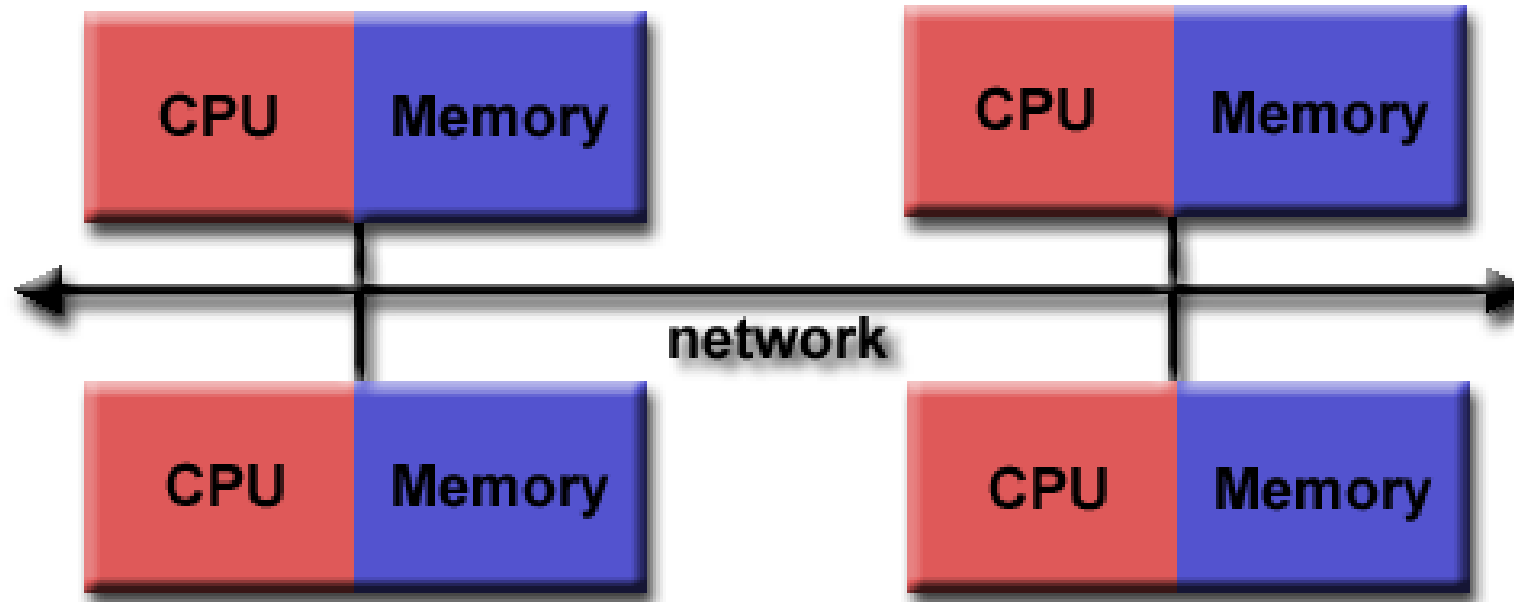


# Arquiteturas de Memória

- Vantagens da memória compartilhada
  - Para o programador é mais simples programar uma memória compartilhada pois não é preciso se preocupar com o acesso
  - O compartilhamento de informações é mais rápido
- Desvantagens da memória compartilhada
  - O modelo raramente é escalável, pois mais CPUs irão criar um problema de gerenciamento
  - A sincronização do acesso fica a cargo do programador

# Arquiteturas de memória

- Memória distribuída

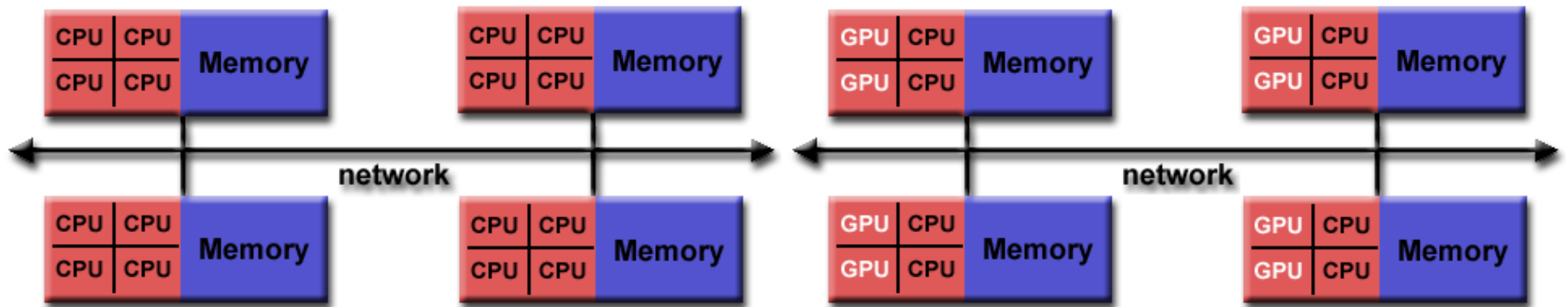


# Arquiteturas de Memória

- Vantagens da memória distribuída
  - Escalável
  - Acesso rápido por parte do processador “dono” da memória
  - Custo de hardware padrão
- Desvantagens da memória compartilhada
  - O programador terá que codificar a comunicação entre os processadores
  - Quanto maior o número de elementos de processamento mais complexo o mapeamento do sistema
  - Tempo de acesso difícil de controlar, não-uniforme

# Arquiteturas de memória

- Memória híbrida
  - Modelo mais utilizado atualmente
  - Cada máquina sabe de sua própria memória apenas
  - Comunicação pela rede



# Bibliografia

- PACHECO, P. An Introduction to Parallel Programming. Morgan Kaufmann, Burlington. 391p. 2011.
- [https://computing.llnl.gov/tutorials/parallel\\_comp/#HybridMemory](https://computing.llnl.gov/tutorials/parallel_comp/#HybridMemory)
- SCHIMIDT, B., GONZÁLEZ-DOMINGUEZ, J., HUNDT, C., SCHLARB, M. Parallel Programming: Concepts and Practice. Morgan Kaufmann, Cambridge. 405p. 2018.



# Programação Paralela: das *threads* aos FPGAs

## Conceitos e Terminologia

Prof. Ricardo Menotti  
[menotti@ufscar.br](mailto:menotti@ufscar.br)

Prof. Maurício Acconcia Dias  
[macccdias@gmail.com](mailto:macccdias@gmail.com)

Prof. Helio Crestana Guardia  
[helio.guardia@ufscar.br](mailto:helio.guardia@ufscar.br)

Departamento de Computação  
Universidade Federal de São Carlos

Atualizado em: 10 de maio de 2020

