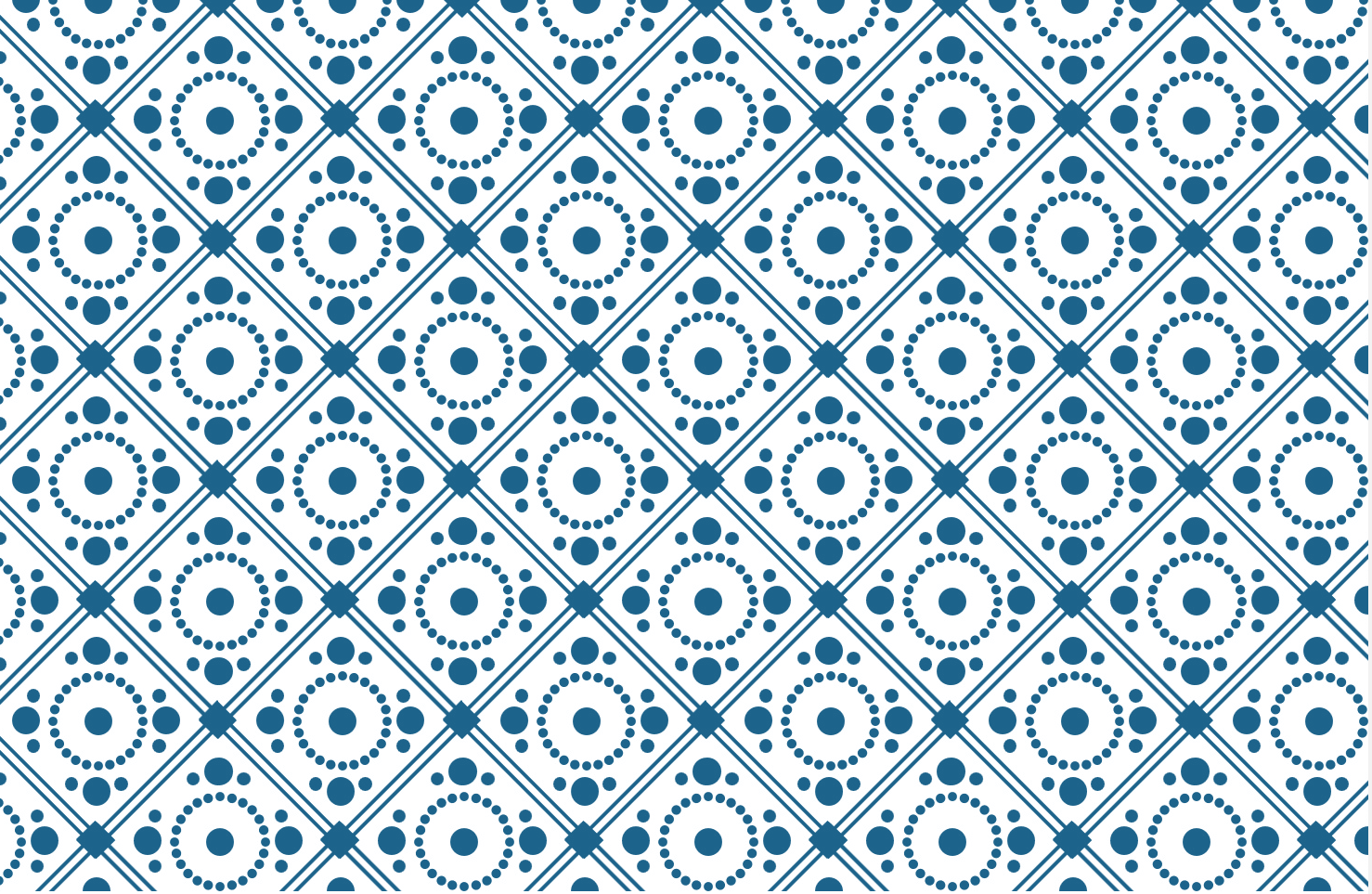
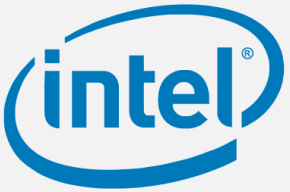




INTEL MODERN CODE PARTNER APRESENTAÇÃO

Slides baseados no material de Marco A. Zanata Alves - UFPR (2016)
Curso: CI316 – Programação Paralela

O QUE ESPERAR DESSE WORKSHOP

PROJETO INTEL MCP

Projeto com a Intel com foco principal em organização de workshops sobre programação paralela moderna com OpenMP

Discutir sobre oportunidades de execução paralela em termos de desempenho e eficiência energética

Explorar os desafios da área, lei de Amdahl, falso compartilhamento, overhead, etc.

EQUIPE

Philippe Olivier Alexandre Navaux – GPPD - UFRGS

Marco A. Zanata Alves – LSE - UFPR

Matthias Diener – GPPD - UFRGS

Eduardo H. Molina da Cruz – GPPD - UFRGS

Demais membros

(mestrandos e doutorandos)

GPPD – UFRGS

PROCEDIMENTOS DIDÁTICOS

4 Aulas Expositivas / Práticas

1ª Aula – Introdução sobre computação paralela e motivação

2ª Aula – Hello World e como as threads funcionam

3ª Aula – Funcionalidades básicas do OpenMP

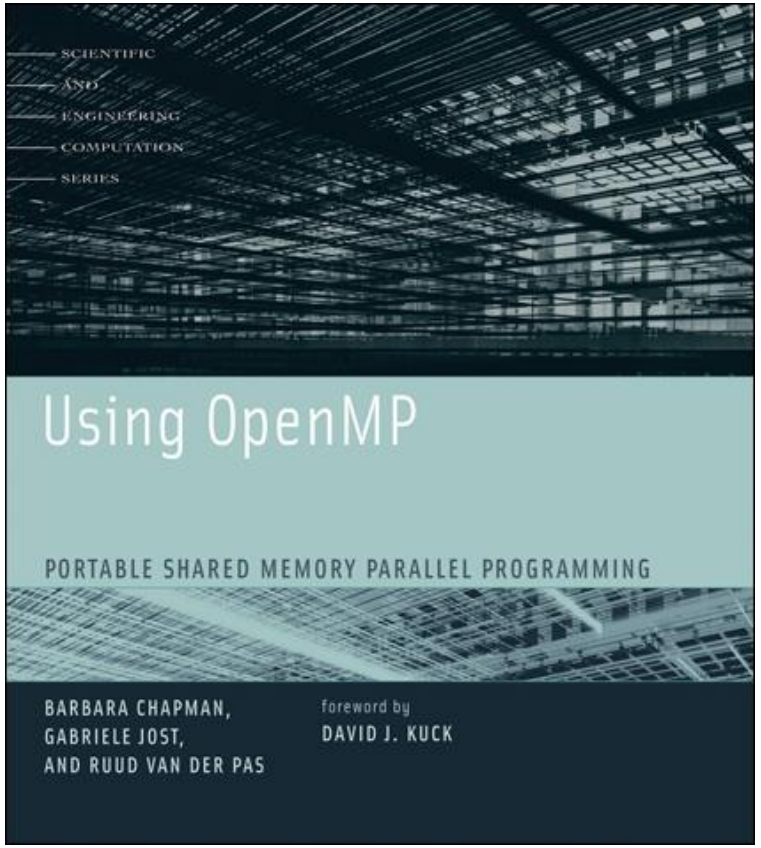
4ª Aula – Trabalhando com OpenMP e tópicos avançados

} **Active learning**

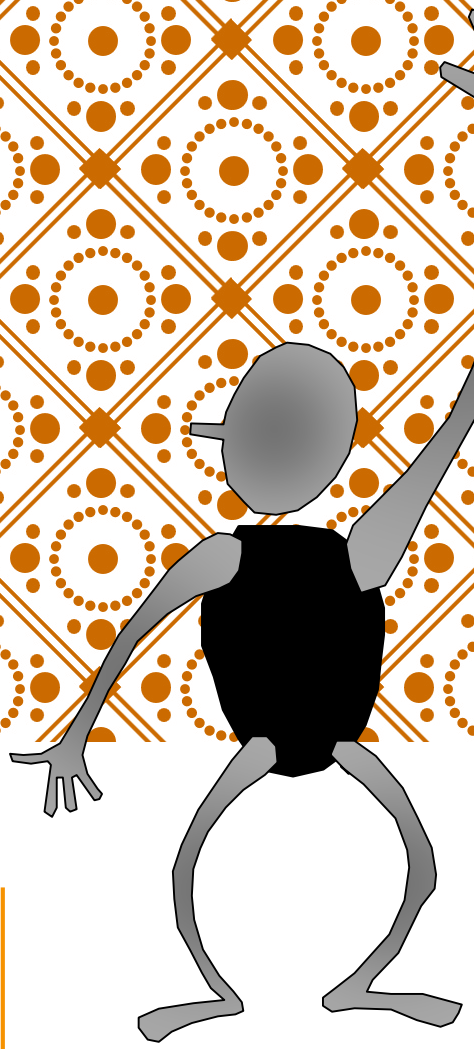
Estamos assumindo que você está confortável em programar em linguagem C

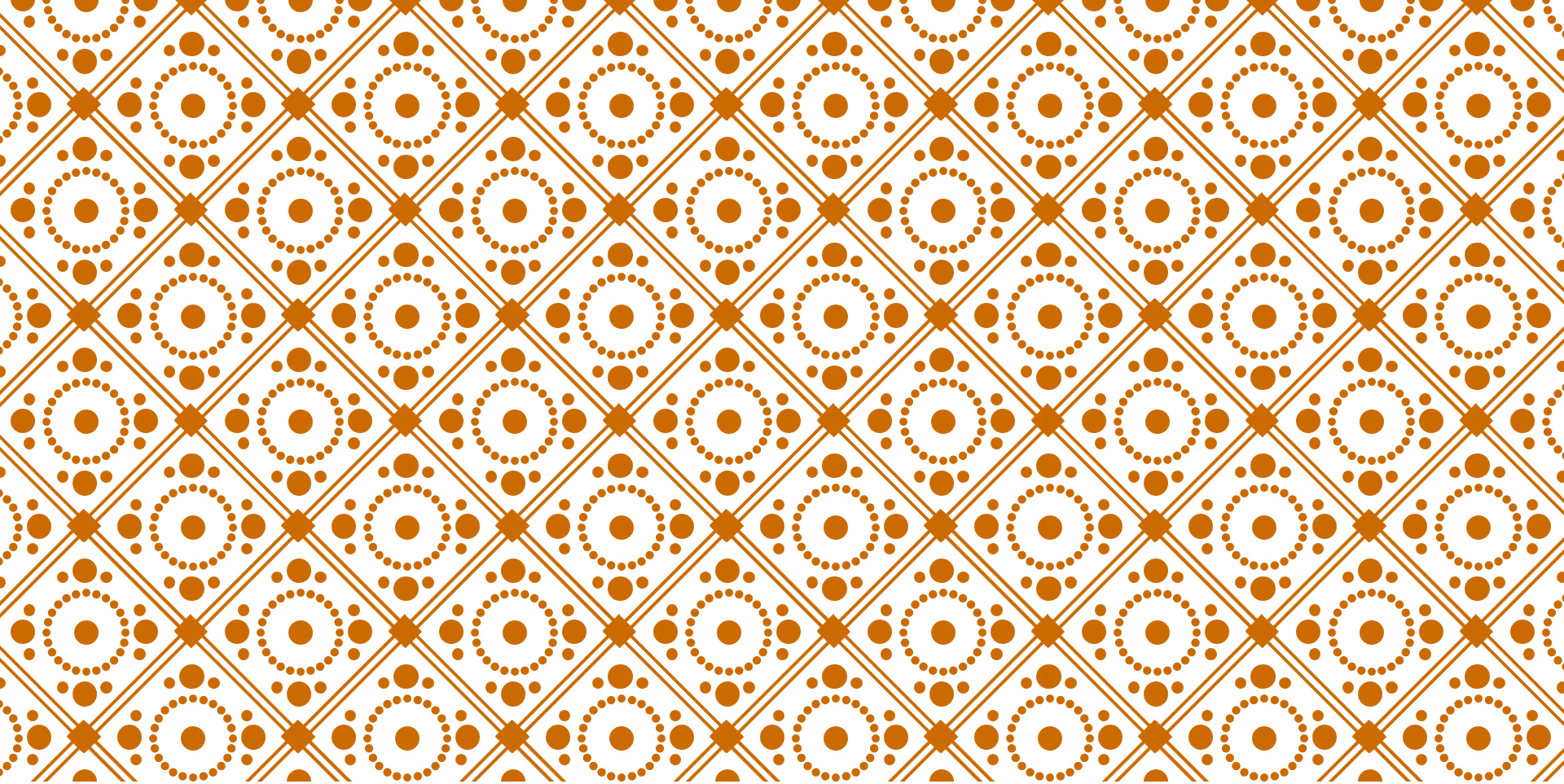
BIBLIOGRAFIA BÁSICA

Using OpenMP – Portable Shared Memory Parallel Programming
Autores: Barbara Chapman, Gabriele Jost and Ruud van der Pas
Editora: MIT Press
Ano: 2007



NÃO TENHA MEDO DE PERGUNTAR!!!





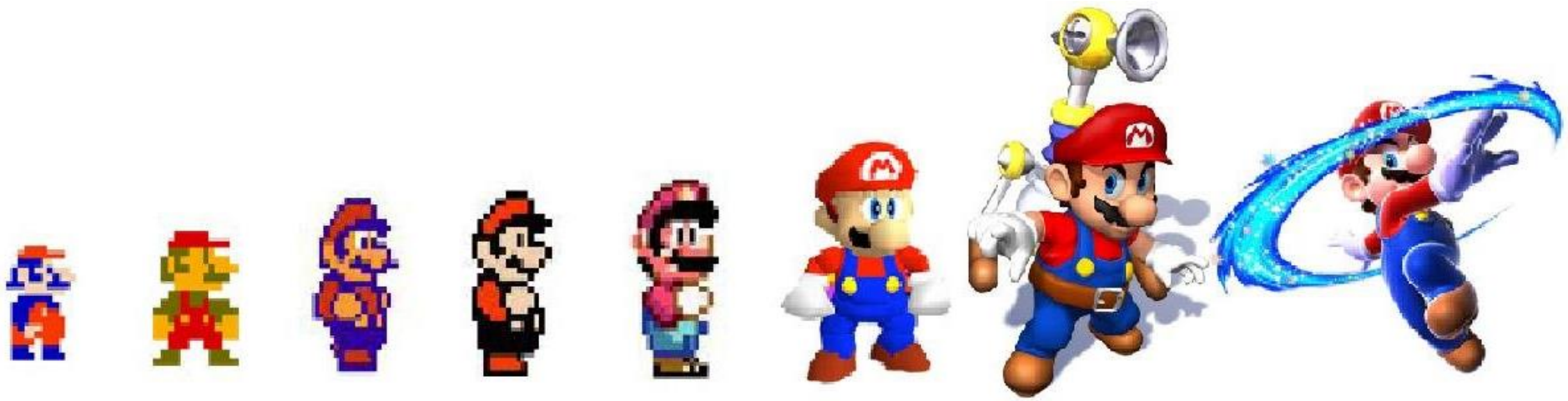
APRESENTAÇÃO DA ÁREA

POR QUE ESTUDAR PROGRAMAÇÃO PARALELA?

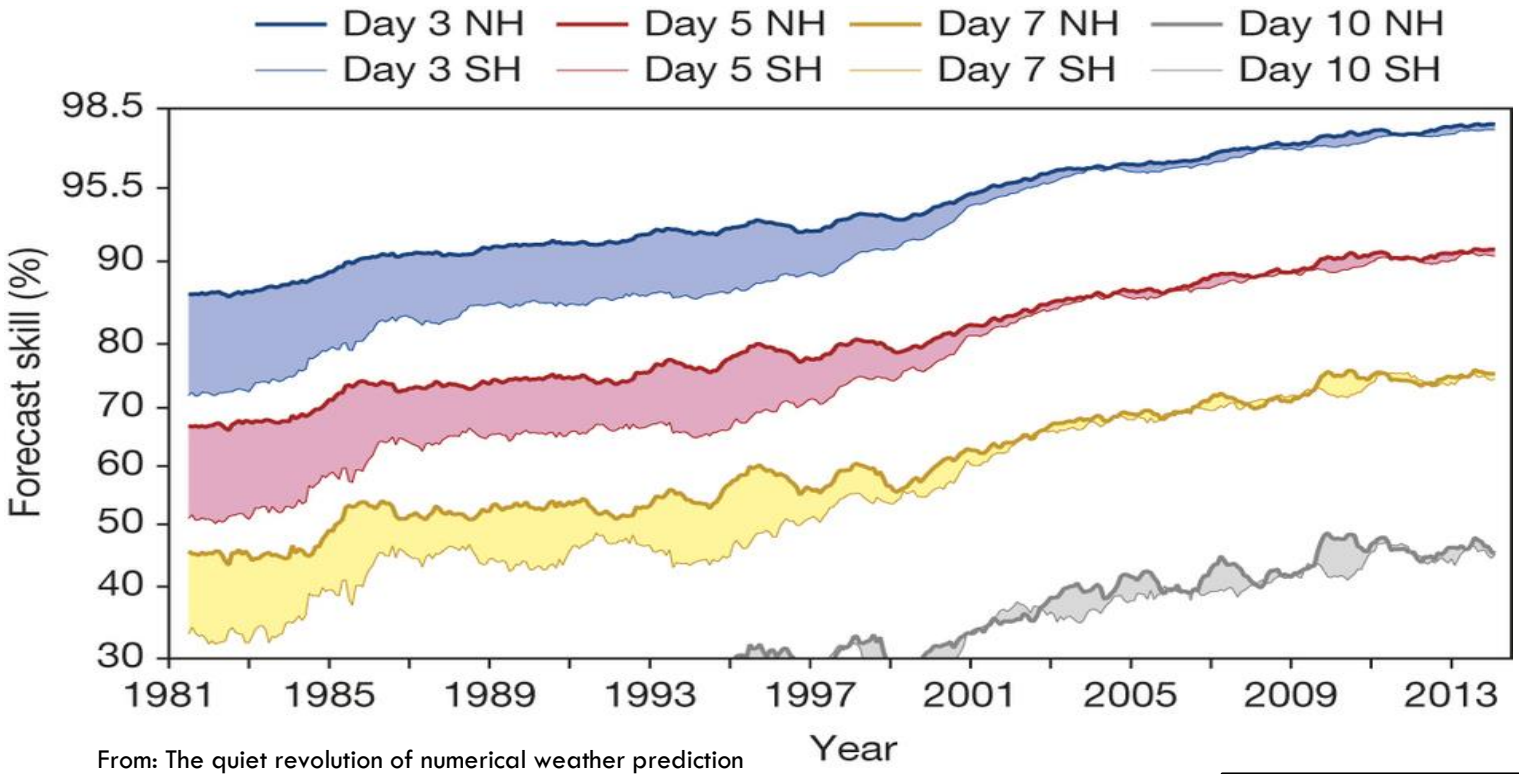
Os programas já não são rápidos o suficiente?

As máquinas já não são rápidas o suficiente?

REQUISITOS SEMPRE MUDANDO



REQUISITOS SEMPRE MUDANDO

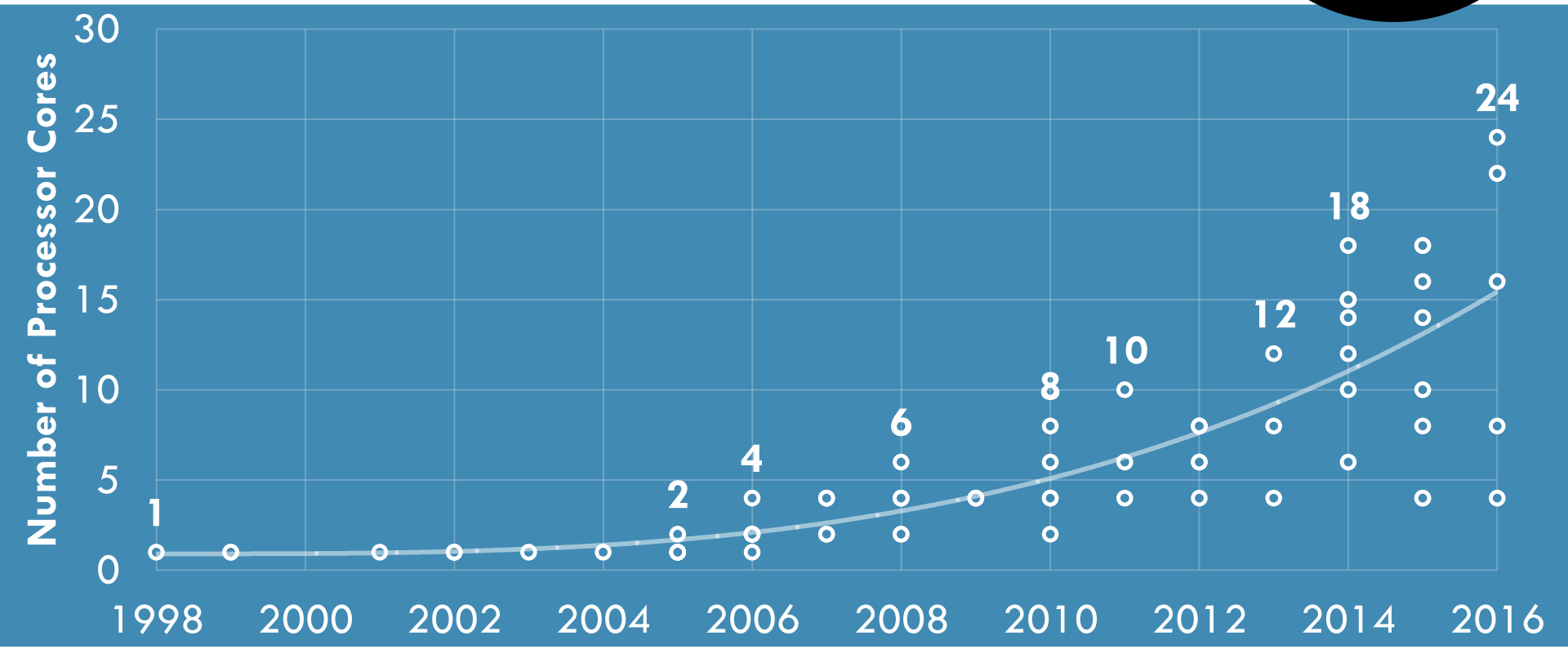


From: The quiet revolution of numerical weather prediction
Peter Bauer, Alan Thorpe & Gilbert Brunet - Nature *525*, 47–55 (03 September 2015)

NH = North Hemisphere
SH = South Hemisphere

EVOLUÇÃO DO INTEL XEON

Onde comprar um processador single-core?



PORQUÊ PROGRAMAÇÃO PARALELA?

Dois dos principais motivos para utilizar programação paralela são:

- **Reduzir o tempo** necessário para solucionar um problema.
- **Resolver problemas mais complexos** e de maior dimensão.

Outros motivos são: ???

PORQUÊ PROGRAMAÇÃO PARALELA?

Dois dos principais motivos para utilizar programação paralela são:

- **Reduzir o tempo** necessário para solucionar um problema.
- **Resolver problemas mais complexos** e de maior dimensão.

Outros motivos são:

- Utilizar recursos computacionais subaproveitados.
- Ultrapassar limitações de memória quando a memória disponível num único computador é insuficiente para a resolução do problema.
- Ultrapassar os limites físicos que atualmente começam a restringir a possibilidade de construção de computadores sequenciais cada vez mais rápidos.

COMO FAZER ATIVIDADES EM PARALELO? NO MUNDO REAL

Você pode enviar cartas/mensagens aos amigos e pedir ajuda

Você pode criar uma lista de tarefas (pool)

Quando um amigo ficar atoa, pegue uma nova tarefa da lista

Você pode ajudar na tarefa ou então ficar apenas gerenciando

Mais amigos, mais tempo para eles chegarem/ se acomodarem

Tarefas curtas ou longas?

Muitos amigos olhando e riscando a lista?

Precisa gerenciar algo mais?

O QUE É NECESSÁRIO?

Definir...

- O que é uma tarefa
- Atividades por tarefa (granularidade)
- Protocolo de trabalho (divisão de tarefas)
- Como retirar uma tarefa da lista
- Onde colocar o resultado final
- Como reportar eventualidades
- Existe ordenação/sincronia entre as tarefas
- Existe dependência/conflito por recursos

OPÇÕES PARA CIENTISTAS DA COMPUTAÇÃO

1. Crie uma **nova linguagem** para programas paralelos
2. Crie um **hardware** para extrair paralelismo
3. Deixe o **compilador** fazer o trabalho sujo
   - Paralelização automática
   - Ou **crie anotações no código sequencial**
4. Use os recursos do **sistema operacional**
   - Com memória compartilhada – threads
   - Com memória distribuída – SPMD
5. Use a **estrutura dos dados** para definir o paralelismo
6. Crie uma **abstração de alto nível** – Objetos, funções aplicáveis, etc.

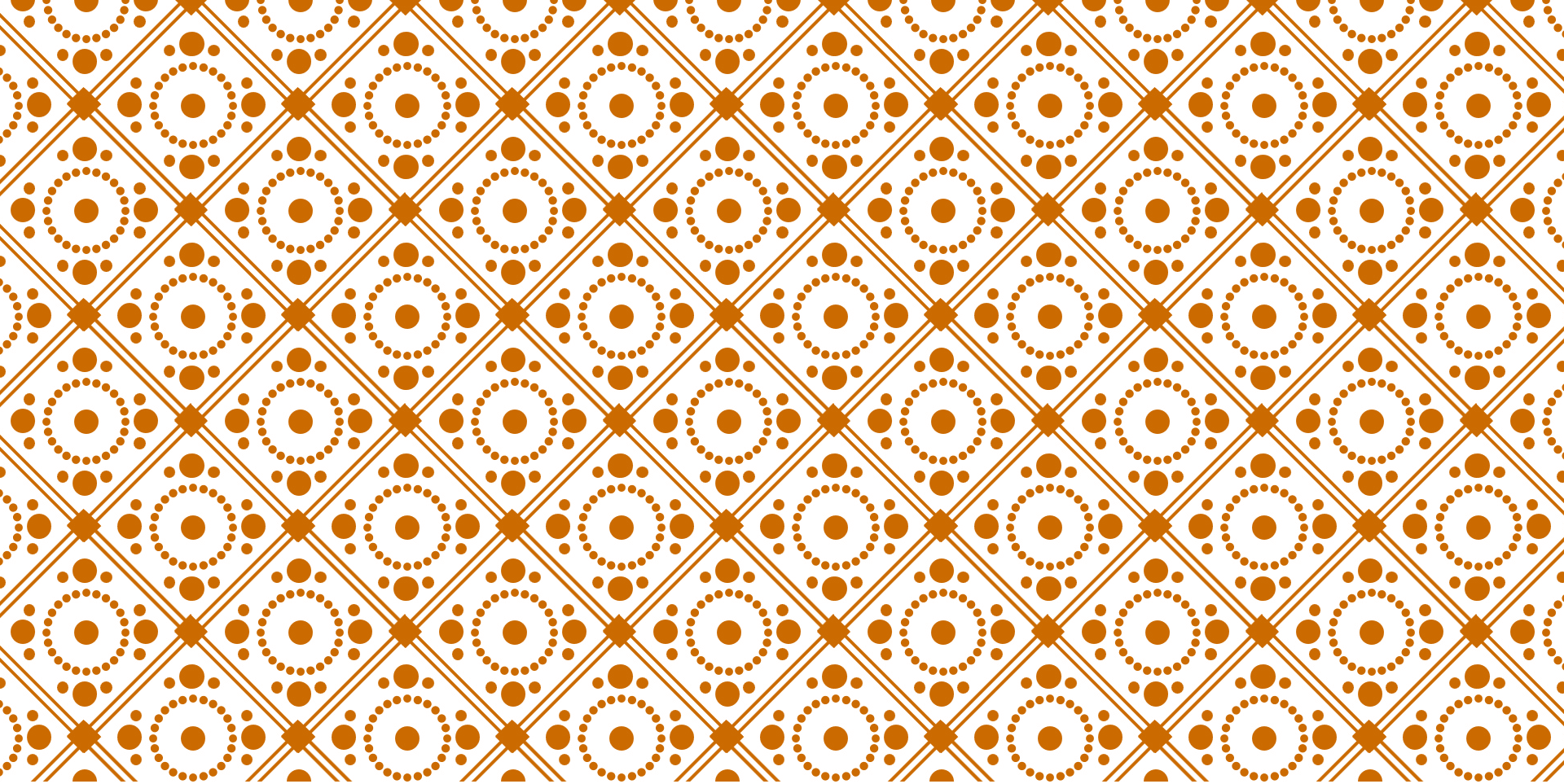
PRINCIPAIS MODELOS DE PROGRAMAÇÃO PARALELA

Programação em Memória Compartilhada (OpenMP, Cilk, CUDA)

- Programação usando processos ou threads.
- Decomposição do domínio ou funcional com granularidade fina, média ou grossa.
- Comunicação através de **memória compartilhada**.
- Sincronização através de mecanismos de exclusão mútua.

Programação em Memória Distribuída (MPI)

- Programação usando processos distribuídos
- Decomposição do domínio com granularidade grossa.
- Comunicação e sincronização por **troca de mensagens**.



METODOLOGIA DE PROGRAMAÇÃO DE FOSTER

METODOLOGIA DE PROGRAMAÇÃO DE FOSTER

Um dos métodos mais conhecidos para desenhar algoritmos paralelos é a metodologia de Ian Foster (1996).

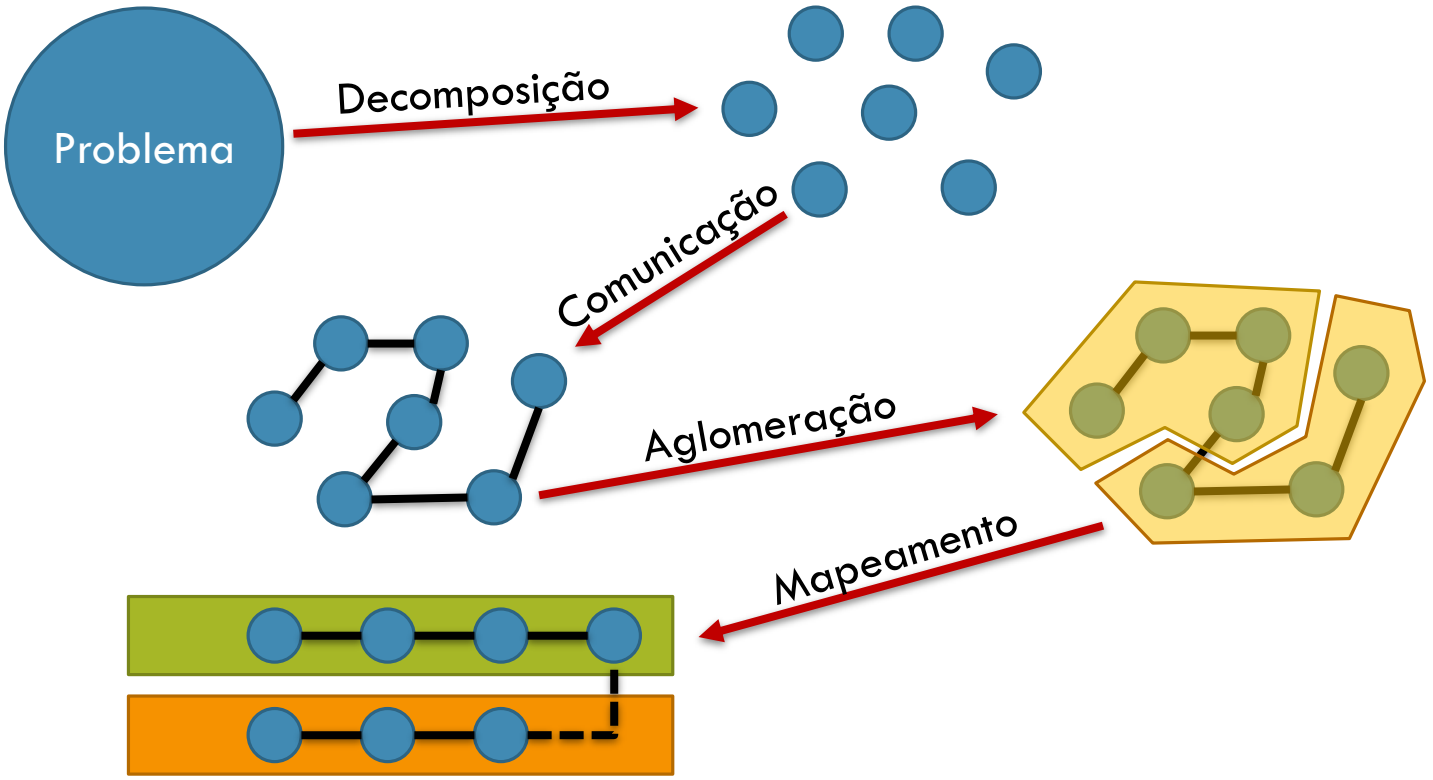
Esta metodologia permite que o programador se concentre inicialmente nos aspectos não-dependentes da arquitetura, como sejam a concorrência e a escalabilidade

Somente depois deve-se considerar os aspectos dependentes da arquitetura, como sejam aumentar a localidade e diminuir a comunicação da computação.

A metodologia de programação de Foster divide-se em 4 etapas:

- Decomposição
- Comunicação
- Aglomeração
- Mapeamento

METODOLOGIA DE PROGRAMAÇÃO DE FOSTER



DECOMPOSIÇÃO

Uma forma de diminuir a complexidade de um problema é conseguir dividi-lo em tarefas mais pequenas de modo a aumentar a concorrência e a localidade de referência de cada tarefa.

Existem duas estratégias principais de decompor um problema:

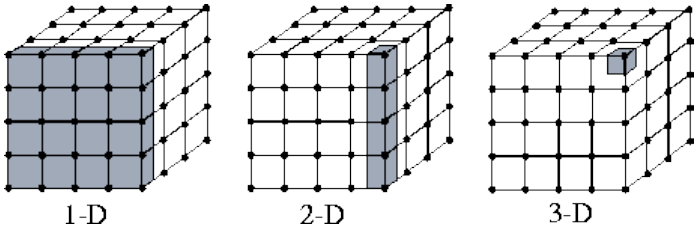
- Decomposição do **Domínio**: decompor o problema em função dos dados.
- Decomposição **Funcional**: decompor o problema em função da computação.

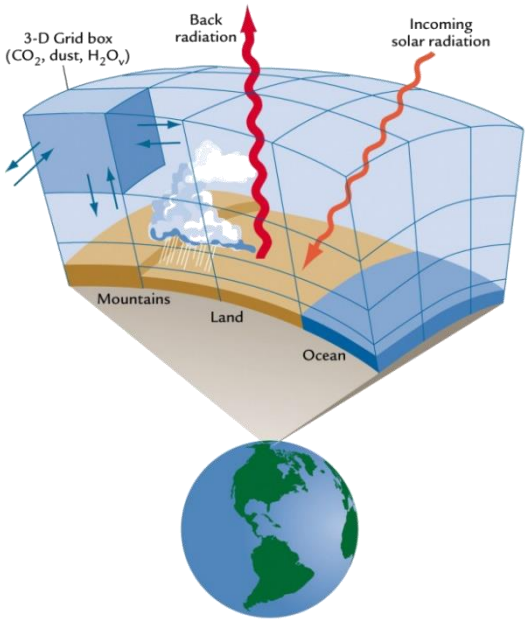
Um boa decomposição tanto divide os dados como a computação em múltiplas tarefas mais pequenas.

DECOMPOSIÇÃO DE DOMÍNIO

Tipo de decomposição em que primeiro se dividem os dados em partições e só depois se determina o processo de associar a computação com as partições.

Todas as tarefas executam as mesmas operações.

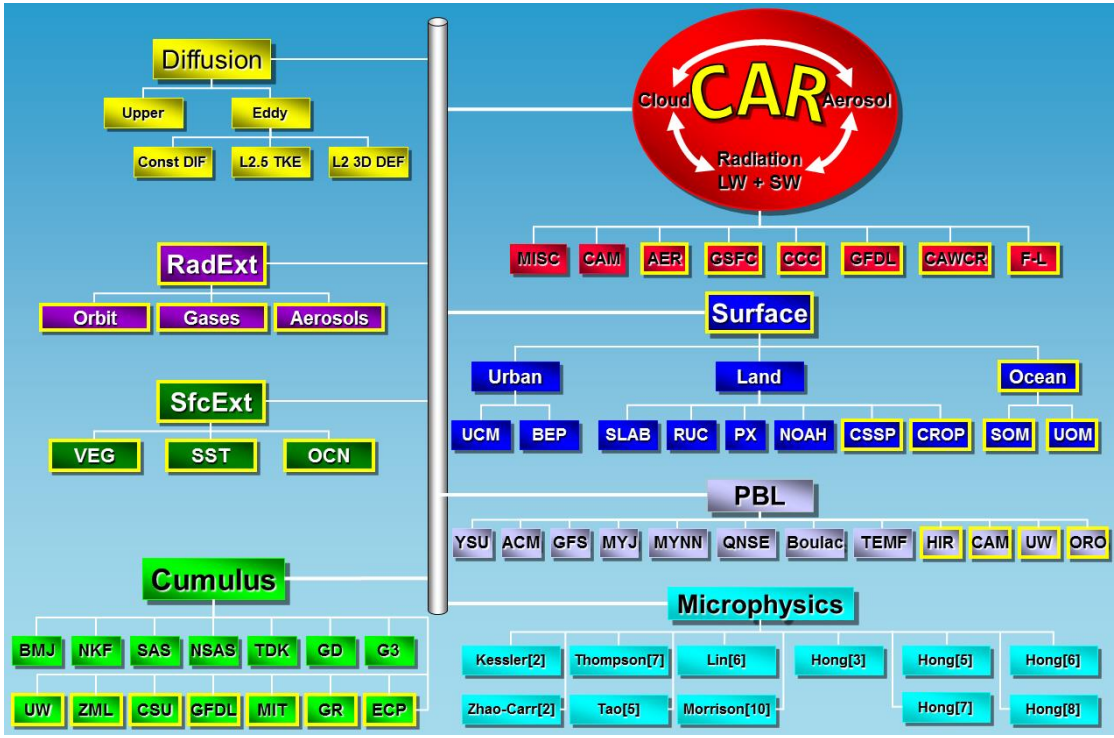




DECOMPOSIÇÃO FUNCIONAL

Tipo de decomposição em que primeiro se divide a computação em partições e só depois se determina o processo de associar os dados com cada partição.

Diferentes tarefas executam diferentes operações.



COMUNICAÇÃO

A natureza do problema e o tipo de decomposição determinam o padrão de **comunicação entre as diferentes tarefas**. A execução de uma tarefa pode envolver a **sincronização** / **acesso a dados** pertencentes/calculados por outras tarefas.

Para haver cooperação entre as tarefas é necessário definir algoritmos e estruturas de dados que permitam uma eficiente troca de informação. Alguns dos principais fatores que limitam essa eficiência são:

COMUNICAÇÃO

A natureza do problema e o tipo de decomposição determinam o padrão de comunicação entre as diferentes tarefas. A execução de uma tarefa pode envolver a sincronização/acesso a dados pertencentes/calculados por outras tarefas.

Para haver cooperação entre as tarefas é necessário definir algoritmos e estruturas de dados que permitam uma eficiente troca de informação.

Alguns dos principais fatores que limitam essa eficiência são:

Latência (tempo mínimo de comunicação entre dois pontos)

Largura de Banda (quantidade de informação comunicada por unidade de tempo)

Espera por sincronização (não contribuem para a computação).

AGLOMERAÇÃO

Aglomeração é o processo de agrupar tarefas em tarefas maiores de modo a diminuir os custos de implementação do algoritmo paralelo e os custos de comunicação entre as tarefas.

Custos de implementação do algoritmo paralelo:

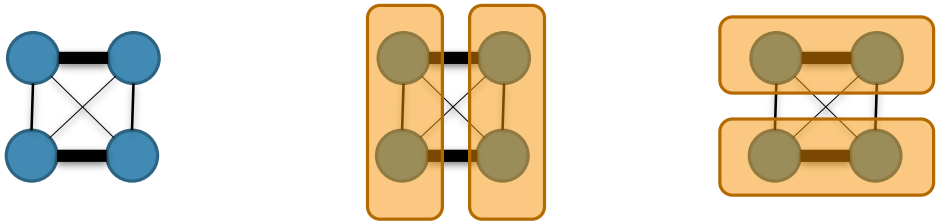
- O agrupamento em tarefas maiores permite uma maior reutilização do código do algoritmo sequencial na implementação do algoritmo paralelo.
- No entanto, o agrupamento em tarefas maiores deve garantir a escalabilidade do algoritmo paralelo de modo a evitar posteriores alterações (e.g. aglomerar as duas últimas dimensões duma matriz de dimensão 8×128×256 restringe a escalabilidade a um máximo de 8 processadores).

AGLOMERAÇÃO

Custos de comunicação entre as tarefas:

- O agrupamento de tarefas elimina os custos de comunicação entre essas tarefas e aumenta a granularidade da computação.

- O agrupamento de tarefas com pequenas comunicações individuais em tarefas com comunicações maiores permite aumentar a granularidade das comunicações e reduzir o número total de comunicações.



GRANULARIDADE

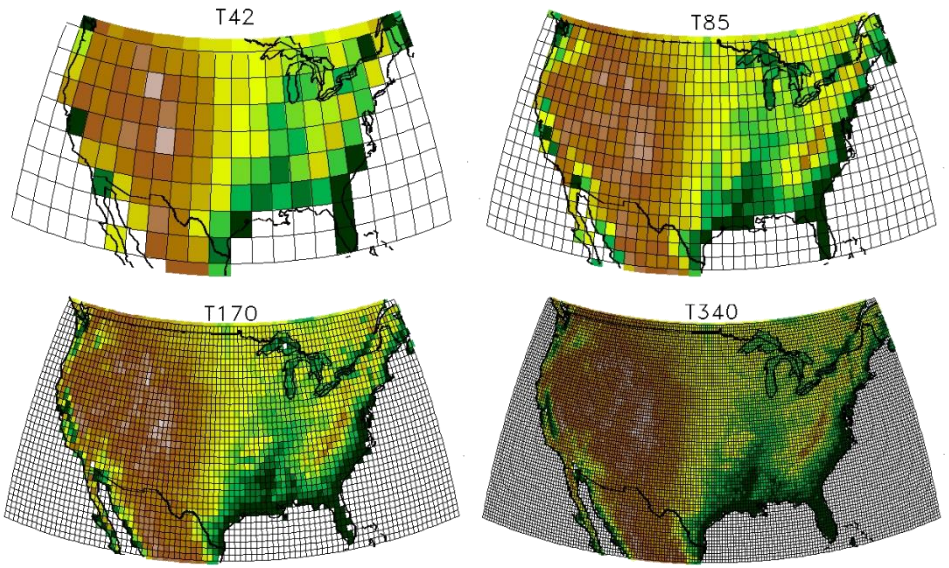
“Como agrupar a computação de modo a obter o máximo desempenho?”

Períodos de computação são tipicamente separados por períodos de comunicação entre as tarefas.

A granularidade é a medida qualitativa da razão entre computação e comunicação.

O número e o tamanho das tarefas em que a computação é agrupada determina a sua granularidade.

A granularidade pode ser **fina**, **média** ou **grossa**.



GRANULARIDADE

Granularidade Fina - Grande número de pequenas tarefas.

- A razão entre computação e comunicação é baixo.
- (+) Fácil de conseguir um balanceamento de carga eficiente.
- (–) O tempo de computação de uma tarefa nem sempre compensa os custos de criação, comunicação e sincronização.
- (–) Difícil de se conseguir melhorar o desempenho.

Granularidade Grossa - Pequeno número de grandes tarefas.

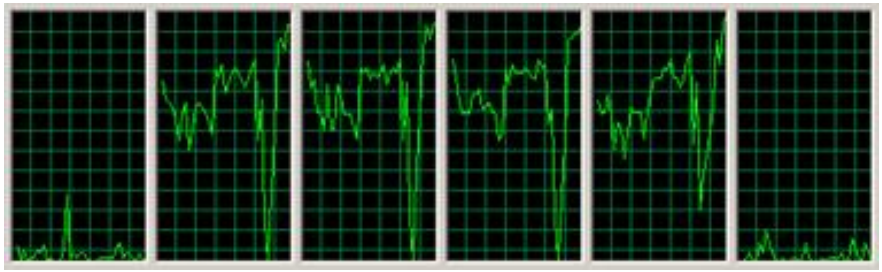
- A razão entre computação e comunicação é grande.
- (–) Difícil de conseguir um balanceamento de carga eficiente.
- (+) O tempo de computação compensa os custos de criação, comunicação e sincronização.
- (+) Oportunidades para se conseguir melhorar o desempenho.

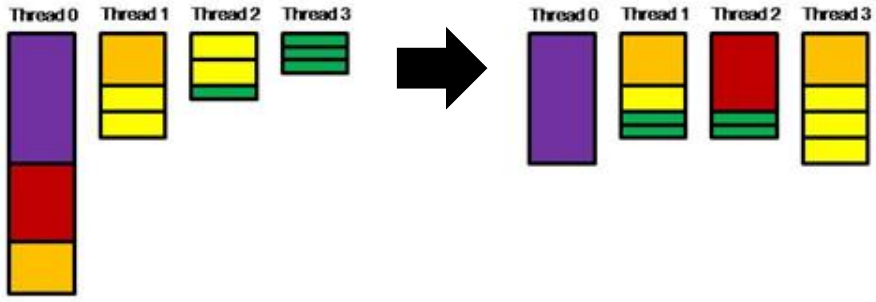
MAPEAMENTO E BALANCEAMENTO DE CARGA

O balanceamento de carga refere-se à capacidade de distribuir tarefas pelos processadores de modo a que todos os processadores estejam ocupados todo o tempo.

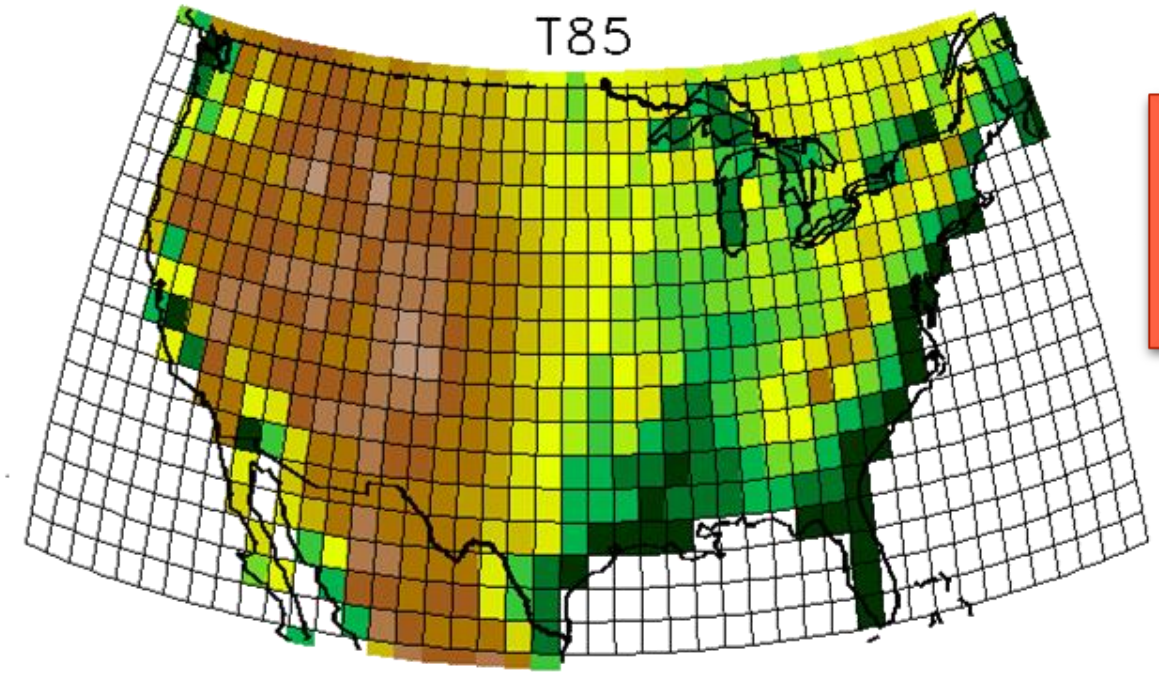
O balanceamento de carga pode ser visto como uma função de minimização do tempo em que os processadores não estão ocupados.

O balanceamento de carga pode ser estático (em tempo de compilação) ou dinâmico (em tempo de execução).



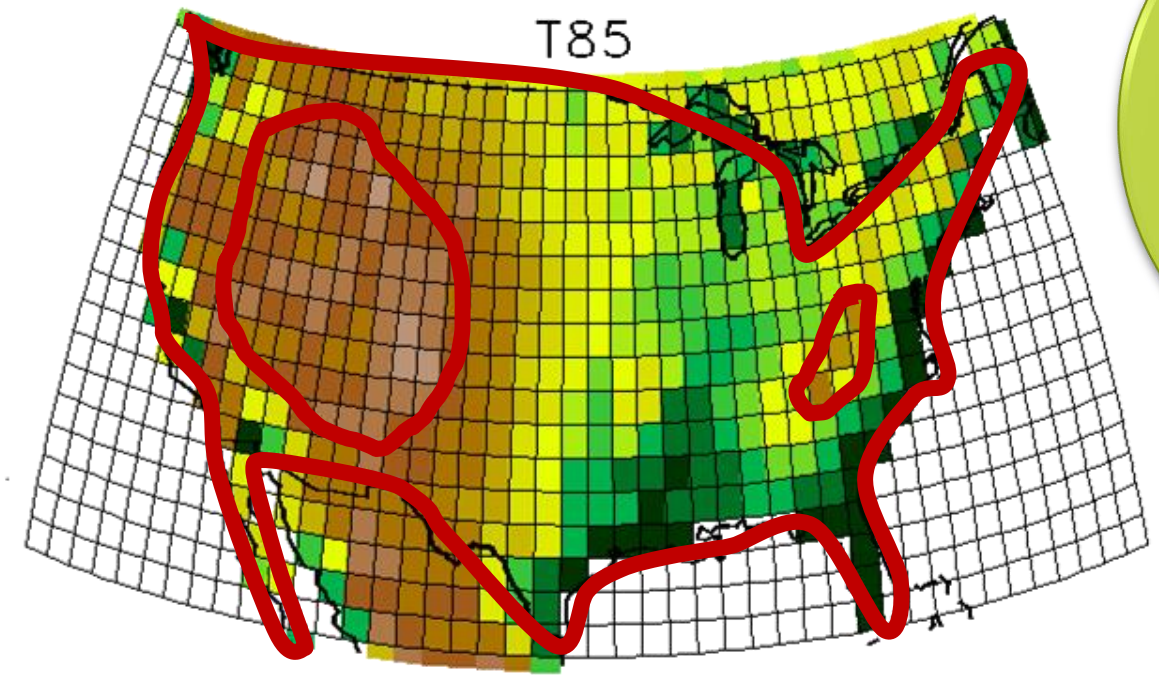


BALANCEAMENTO DE CARGA



Como dividir esse problema para evitar desbalanceamento de carga?

BALANCEAMENTO DE CARGA



Precisamos entender bem o problema!

Continente ou Sol ou Calor ➔
+ trabalho de simulação

FATORES DE LIMITAÇÃO DO DESEMPENHO

Código Sequencial: existem partes do código que são inerentemente sequenciais (e.g. iniciar/terminar a computação).

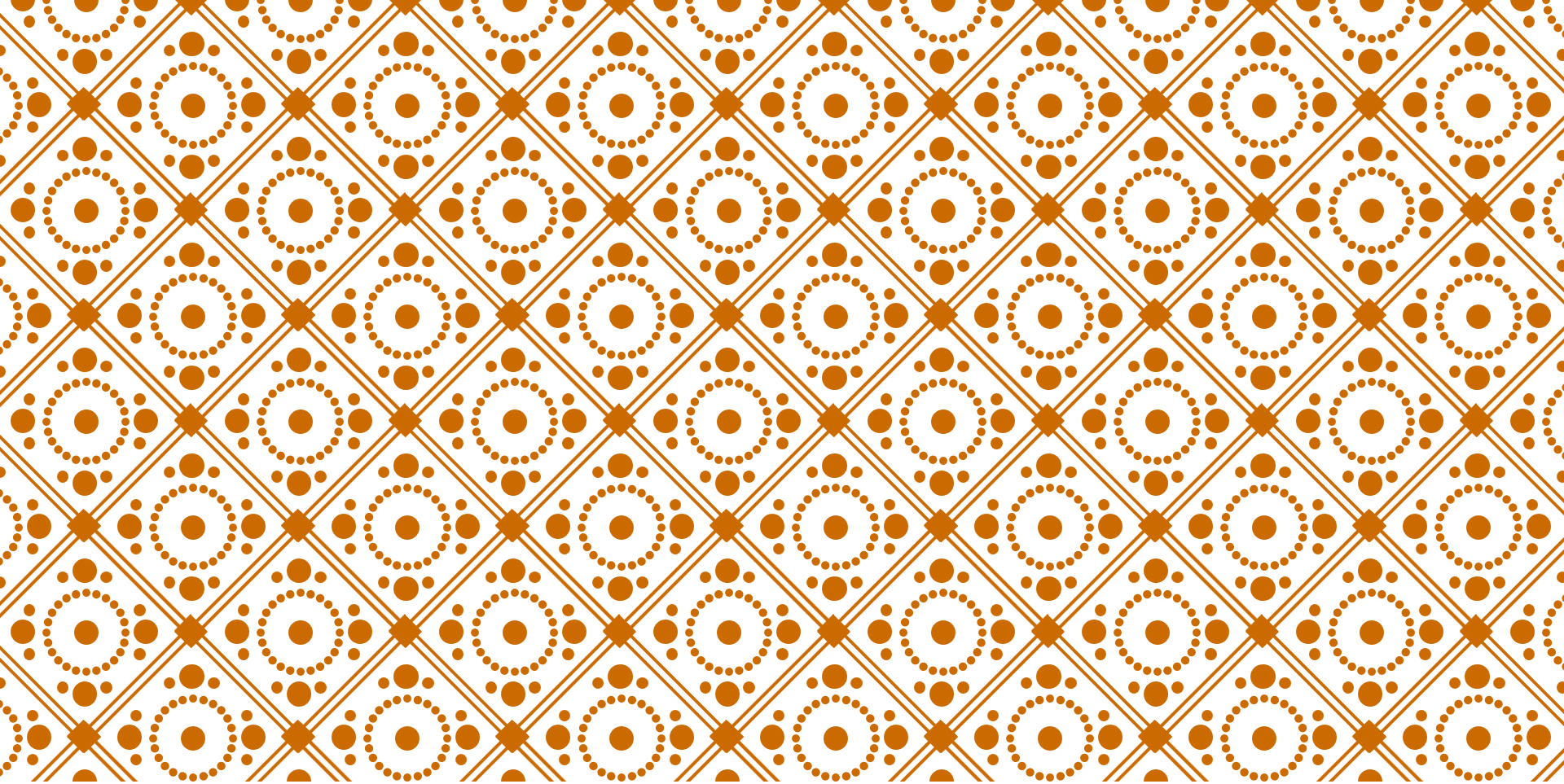
Concorrência/Paralelismo: o número de tarefas pode ser escasso e/ou de difícil definição.

Comunicação: existe sempre um custo associado à troca de informação e enquanto as tarefas processam essa informação não contribuem para a computação.

Sincronização: a partilha de dados entre as várias tarefas pode levar a problemas de contenção no acesso à memória e enquanto as tarefas ficam à espera de sincronizar não contribuem para a computação.

Granularidade: o número e o tamanho das tarefas é importante porque o tempo que demoram a ser executadas tem de compensar os custos da execução em paralelo (e.g. custos de criação, comunicação e sincronização).

Balanceamento de Carga: ter os processadores maioritariamente ocupados durante toda a execução é decisivo para o desempenho global do sistema.



MÉTRICAS DE DESEMPENHO

MÉTRICAS DE DESEMPENHO PARA APLICAÇÕES PARALELAS

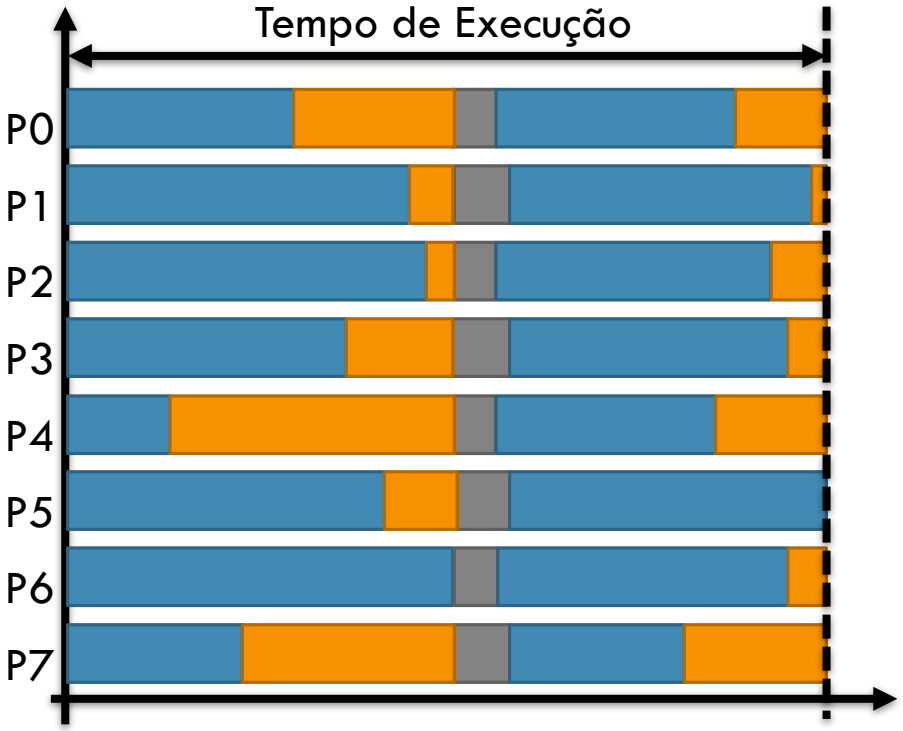
Existem várias medidas que permitem medir/avaliar o desempenho duma aplicação paralela. As mais conhecidas são:

- **Speedup** (*), **Overhead**, **Eficiência**, **Custo**
- Isoeficiência, Scaled speedup, Redundância, Utilização, Qualidade, Métrica de Karp-Flatt, outras

Existem igualmente várias leis/métricas que permitem balizar o comportamento duma aplicação paralela face ao seu potencial desempenho. As mais conhecidas são:

- **Lei de Amdahl**, Lei de Gustafson-Barsis, Lei de Grosch, Conjectura de Minsky, outras

FONTES DE OVERHEAD EM PROGRAMAS PARALELOS

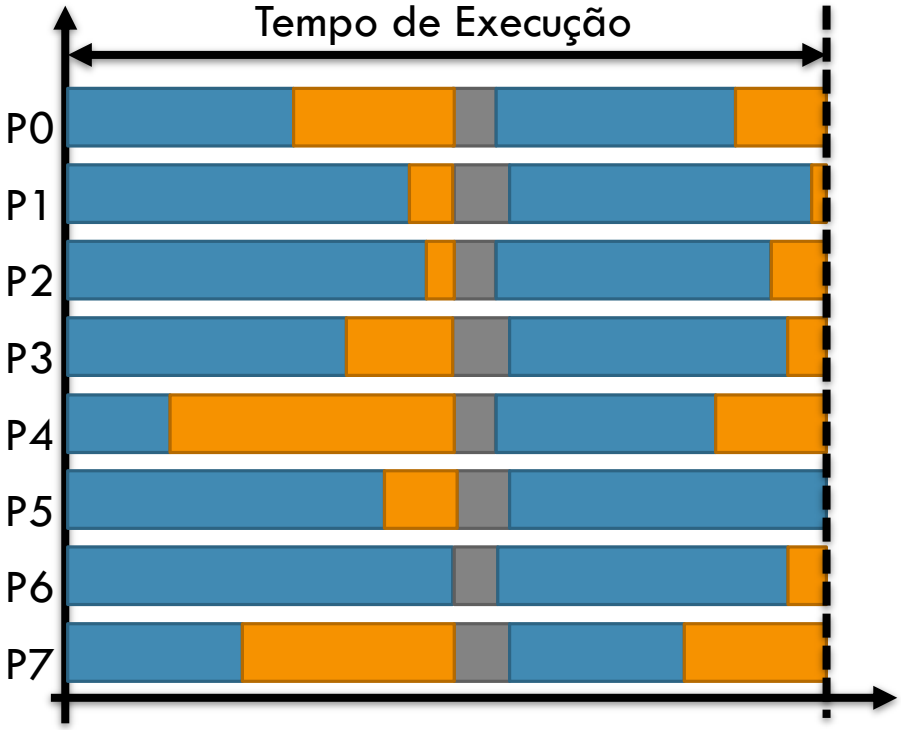


Processo inativo

Comunicação inter-processadores

Computação essencial ou excessiva

FONTES DE OVERHEAD EM PROGRAMAS PARALELOS



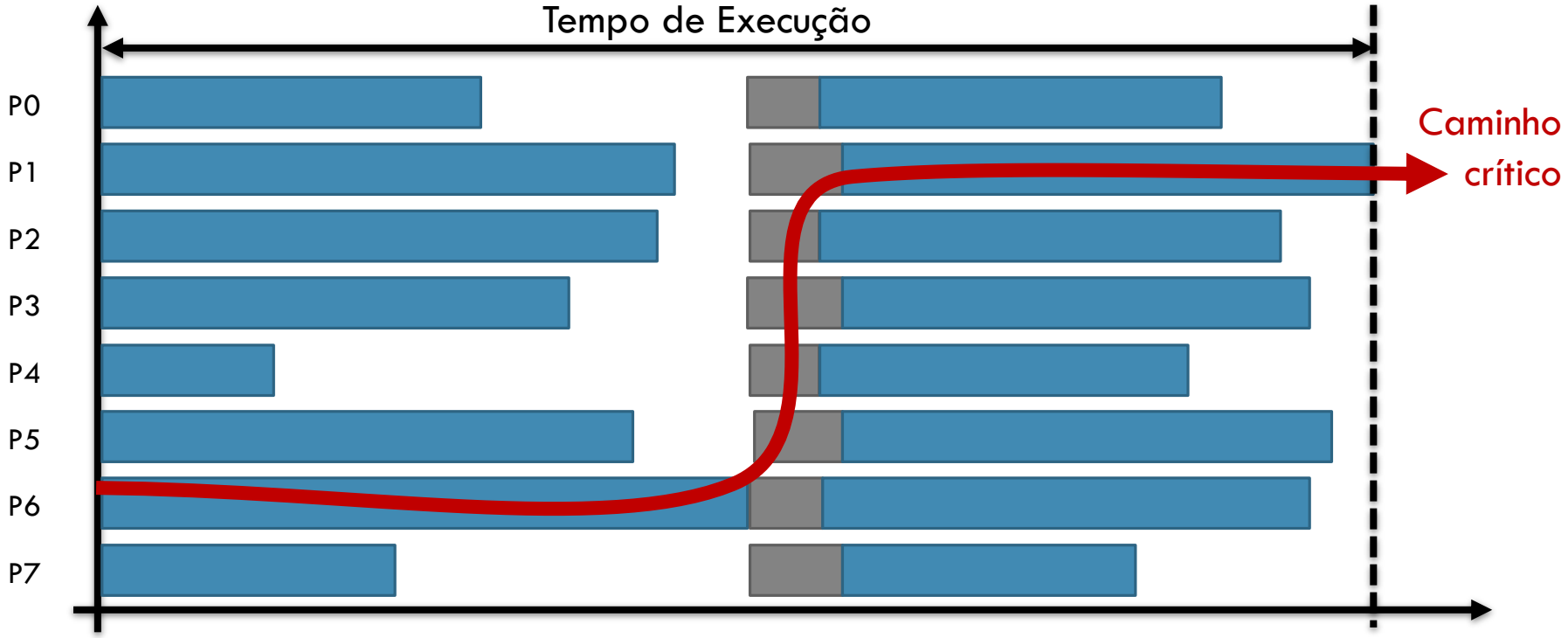
Inativo: Acontece devido a desbalanceamento e/ou sincronização

Comunicação: Trata-se de troca de mensagens ou acesso a dados compartilhados e sincronização

Processamento: O melhor algoritmo sequencial pode ser difícil ou impossível de ser paralelizado.

A diferença de trabalho entre o processor paralelo e o serial é chamado de **overhead**. A inicialização/divisão do trabalho também pode significar overhead.

FONTES DE OVERHEAD EM PROGRAMAS PARALELOS



OVERHEAD

O overhead (T_O) pode ser medido como a diferença entre o tempo sequencial e o a soma de todos os (p) tempos paralelo (com todos os sobre-custos)

$$T_O = p \times T_p - T_s$$

T_s é o tempo de execução do algoritmo sequencial

T_p é o tempo de execução do algoritmo paralelo com p processadores

SPEEDUP

O speedup é uma medida do grau de desempenho. O speedup mede a razão entre o tempo de execução de duas soluções que resolvem o mesmo problema.

$$Speedup = \frac{T_{antigo}}{T_{novo}}$$

T_{antigo} é o tempo de execução de uma solução base

T_{novo} é o tempo de execução da nova solução proposta

SPEEDUP EM COMPUTAÇÃO PARALELA

O speedup é uma medida do grau de desempenho. O speedup mede a razão entre o tempo de execução sequencial e o tempo de execução em paralelo.

$$S(p) = \frac{T_p(1)}{T_p(p)}$$

$T_p(1)$ é o tempo de execução com um processador

$T_p(p)$ é o tempo de execução com p processadores

	1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
T(p)	1000	520	280	160	100
S(p)	**1**	**1,92**	**3,57**	**6,25**	**10,00**

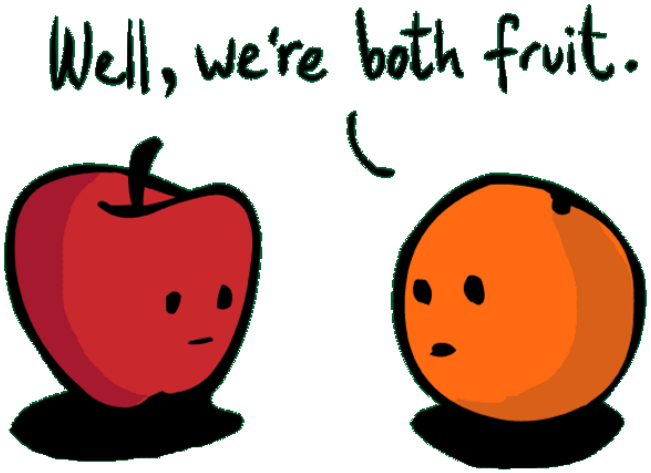
SPEEDUP - ARMADILHAS

Qual o speedup do método de ordenação $T_{odd-even}(4)$ considerando os seguintes tempos:

$T_{bubble-sort}(1) = 150$ seg.

$T_{quick-sort}(1) = 30$ seg.

$T_{odd-even}(1) = 120$ seg. → $\boldsymbol{T_{odd-even}(4) = 40}$ **seg.**



SPEEDUP - ARMADILHAS

Qual o speedup do $T_{odd-even}(4)$ considerando os seguintes tempos:

$T_{bubble-sort}(1) = 150$ seg.

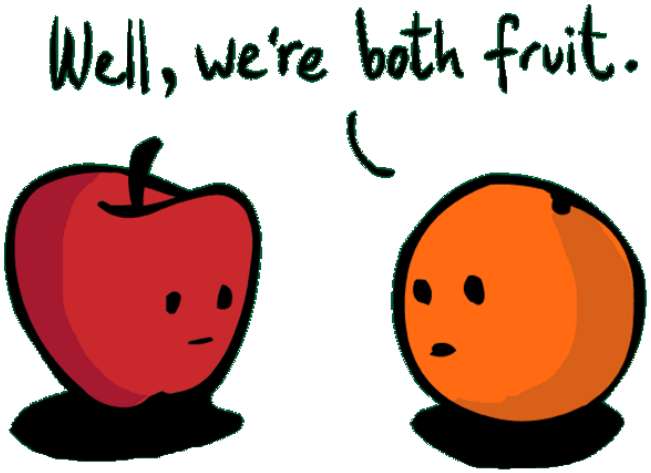
$T_{quick-sort}(1) = 30$ seg.

$T_{odd-even}(1) = 120$ seg. → **$T_{odd-even}(4) = 40$ seg.**

$$S = \frac{T_{bubble-sort}(1)}{T_{odd-even}(4)} = 3.75$$

$$S = \frac{T_{odd-even}(1)}{T_{odd-even}(4)} = 3.00$$

$$S = \frac{T_{quick-sort}(1)}{T_{odd-even}(4)} = 0.75$$



SPEEDUP - ARMADILHAS

Qual o speedup do $T_{odd-even}(4)$ considerando os seguintes tempos:

$T_{bubble-sort}(1) = 150$ seg.

$T_{quick-sort}(1) = 30$ seg.

$T_{odd-even}(1) = 120$ seg. → **$T_{odd-even}(4) = 40$ seg.**

$$S = \frac{T_{bubble-sort}(1)}{T_{odd-even}(4)} = 3.75$$ (Enganoso)

$$S = \frac{T_{odd-even}(1)}{T_{odd-even}(4)} = 3.00$$ (Justo, compara o mesmo algoritmo)

$$S = \frac{T_{quick-sort}(1)}{T_{odd-even}(4)} = 0.75$$ (Justo, compara com o melhor algoritmo)

SPEEDUP SUPERLINEAR

"Se um único processador consegue resolver um problema em N segundos, podem N processadores resolver o mesmo problema em

$$\text{tempo} \begin{cases} > 1 segundo \\ = 1\ segundo \\ < 1\ segundo \end{cases}$$?"

SPEEDUP SUPERLINEAR

O speedup diz-se superlinear quando a razão entre o tempo de execução sequencial e o tempo paralelo com p processadores é maior do que p.

$$\frac{T(1)}{T(p)} \geq \mathrm{p}$$

Alguns dos fatores que podem fazer com que o speedup seja superlinear são: ???

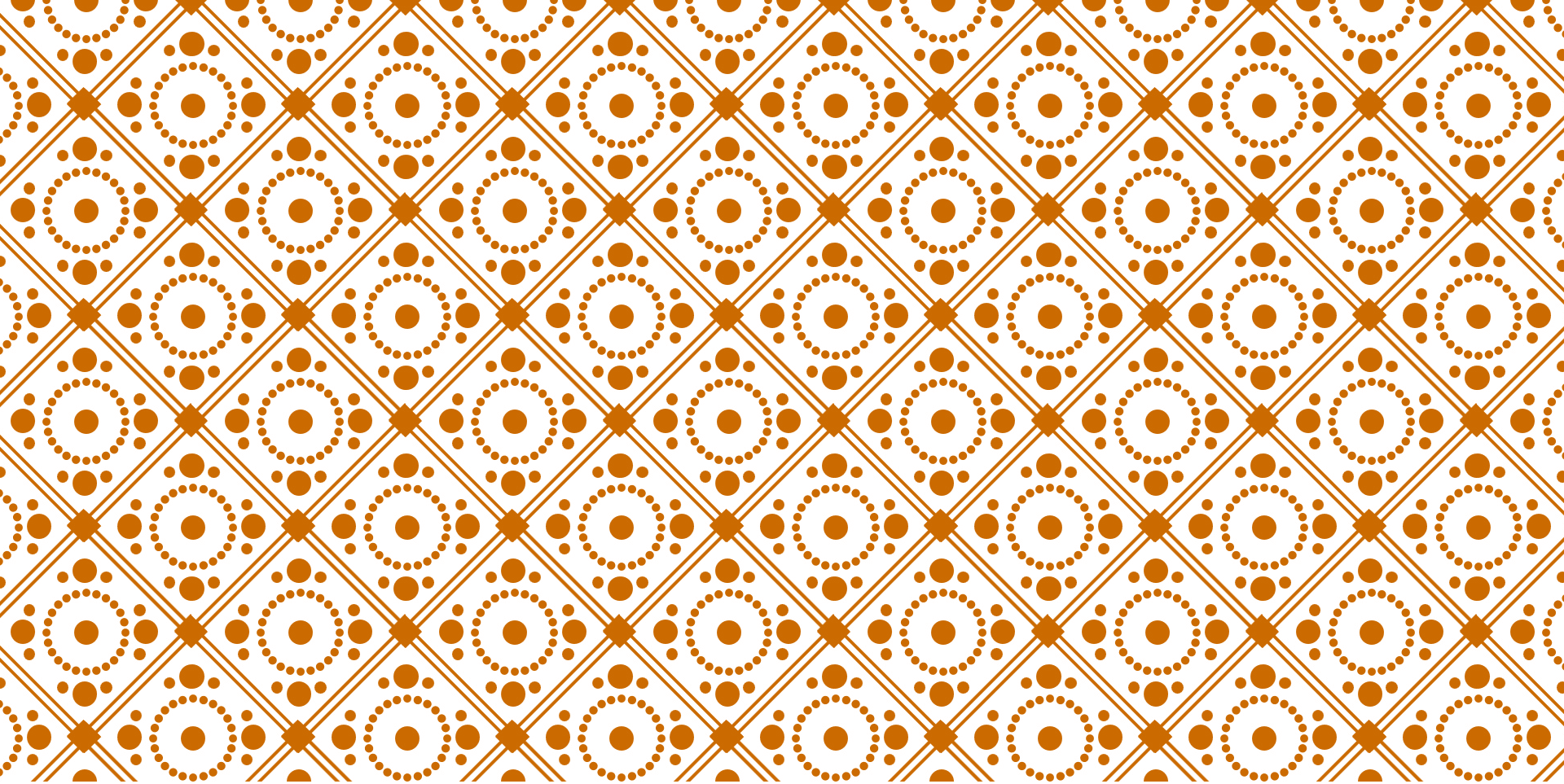
SPEEDUP SUPERLINEAR

O speedup diz-se superlinear quando a razão entre o tempo de execução sequencial e o tempo paralelo com p processadores é maior do que p.

$$\frac{T(1)}{T(p)} \geq \mathrm{p}$$

Alguns dos fatores que podem fazer com que o speedup seja superlinear são:

- Aumento da capacidade de memória (o problema passa a caber todo em memória).
- Subdivisão do problema (tarefas menores geram menos cache misses).
- Aleatoriedade da computação em problemas de optimização ou com múltiplas soluções (ex. achar solução para o problema de n-rainhas).



EFICIÊNCIA E ESCALABILIDADE

EFICIÊNCIA

A eficiência é uma medida do grau de aproveitamento dos recursos computacionais. A eficiência mede a razão entre o grau de desempenho e os recursos computacionais disponíveis.

$$\mathrm{E}(p) = \frac{S(p)}{p} = \frac{T_p(1)}{p \times T_p(p)}$$

S(p) é o speedup para p processadores

	1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
T(p)	1000	520	280	160	100
S(p)	1	1,92	3,57	6,25	10,00
E(p)	**1**	**0,96**	**0,89**	**0,78**	**0,63**

EFICIÊNCIA E ESCALABILIDADE

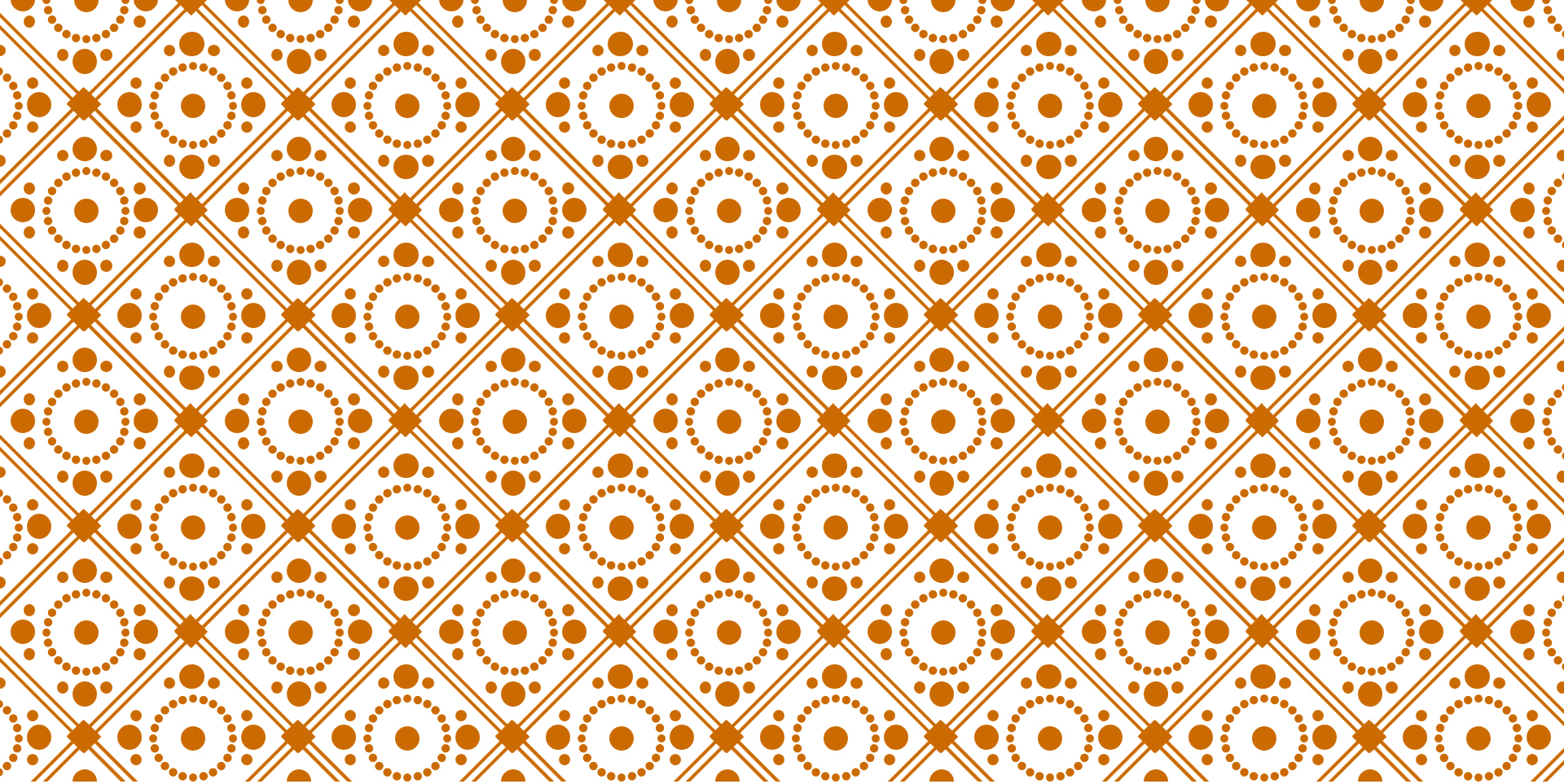
Uma aplicação é dita de **fortemente escalável** quando demonstra a capacidade de manter a mesma eficiência à medida que o número de processadores aumenta.

Uma aplicação é dita de **fracamente escalável** quando demonstra a capacidade de manter a mesma eficiência à medida que o número de processadores e a dimensão do problema aumentam proporcionalmente.

Uma aplicação é dita **não escalável** quando não apresenta capacidade de manter eficiência.

$$\text{Eficiência}(p) = \frac{S(p)}{p}$$

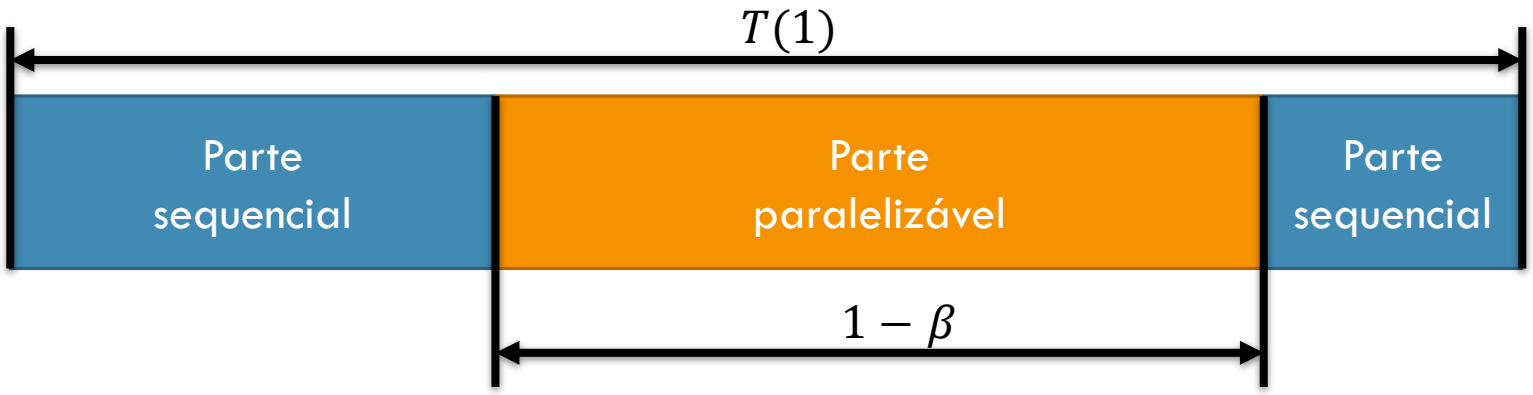
A escalabilidade de uma aplicação reflete a sua capacidade de utilizar mais recursos computacionais de forma efetiva.



LEI DE AMDAHL (1967)

LEI DE AMDAHL

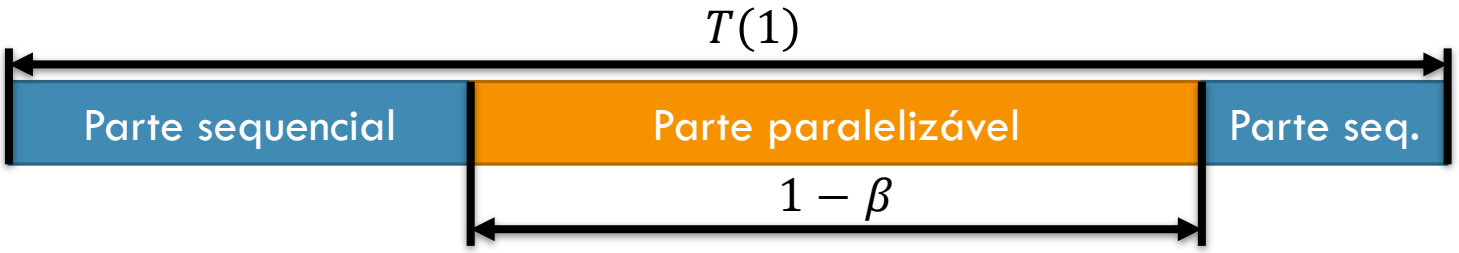
Tempo de execução em um único processador:



$\beta =$ fração de código que é **puramente sequencial**

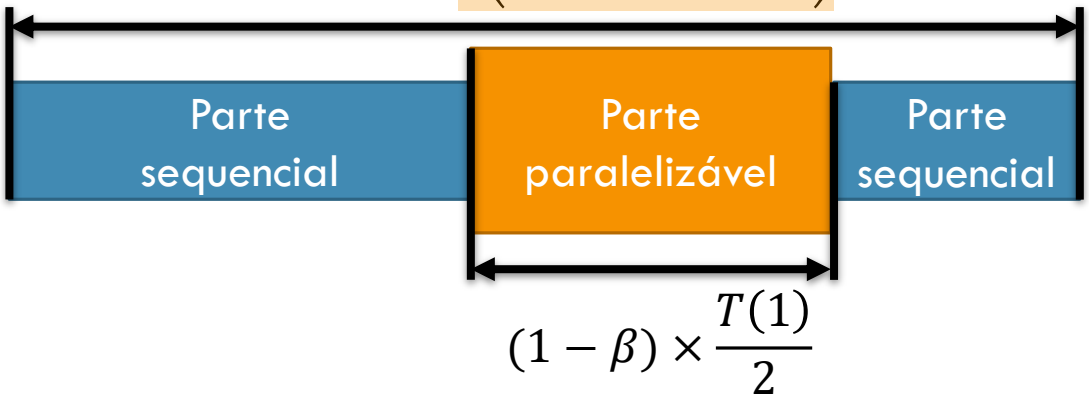
LEI DE AMDAHL

Tempo de execução em um único processador



Tempo de execução em 2 processadores

$$T(2) = T(1) \times \beta + \left((1-\beta) \times \frac{T(1)}{2} \right)$$



LEI DE AMDAHL

Seja $0 \leq \beta \leq 1$ a fração da computação que só pode ser realizada sequencialmente.

A lei de Amdahl diz-nos que o speedup máximo que uma aplicação paralela com p processadores pode obter é:

$$S(p) = \frac{1}{\beta + \frac{(1-\beta)}{p}}$$

A lei de Amdahl também pode ser utilizada para determinar o **limite máximo de speedup** que uma determinada aplicação poderá alcançar independentemente do número de processadores a utilizar (limite máximo teórico).

EXERCÍCIO: LEI DE AMDAHL

Suponha que pretende determinar se é vantajoso desenvolver uma versão paralela de uma determinada aplicação sequencial.

Por experimentação, verificou-se que 90% do tempo de execução é passado em procedimentos que se julga ser possível paralelizar.

Qual é o speedup máximo que se pode alcançar com uma versão paralela do problema executando em 8 processadores?

Qual é o speedup máximo considerando infinitos processadores?

$$S(p) = \frac{1}{\beta + \frac{(1-\beta)}{p}}$$

LEI DE AMDAHL

Suponha que pretende determinar se é vantajoso desenvolver uma versão paralela de uma determinada aplicação sequencial. Por experimentação, verificou-se que 90% do tempo de execução é passado em procedimentos que se julga ser possível paralelizar. Qual é o speedup máximo que se pode alcançar com uma versão paralela do problema executando em 8 processadores?

$$S(p) \leq \frac{1}{0{,}1+\frac{(1-0{,}1)}{8}} \approx 4{,}71$$

Qual é o speedup máximo considerando infinitos processadores?

$$\lim_{p\to\infty}\left(\frac{1}{0{,}1+\frac{(1-0{,}1)}{p}}\right) = 10$$

$p \to \infty$, $\frac{(1-0{,}1)}{p} \to 0$

LIMITAÇÕES DA LEI DE AMDAHL

A lei de Amdahl ignora o custo das operações de comunicação/sincronização associadas à introdução de paralelismo numa aplicação.

Por esse motivo, a lei de Amdahl pode resultar em predições pouco realistas para determinados problemas.

A lei de Amdahl também supõe que o problema se manterá inalterado ao aumentar os recursos computacionais.

Ou seja, considera que a porcentagem puramente sequencial se manterá mesmo modificando o tamanho do problema.