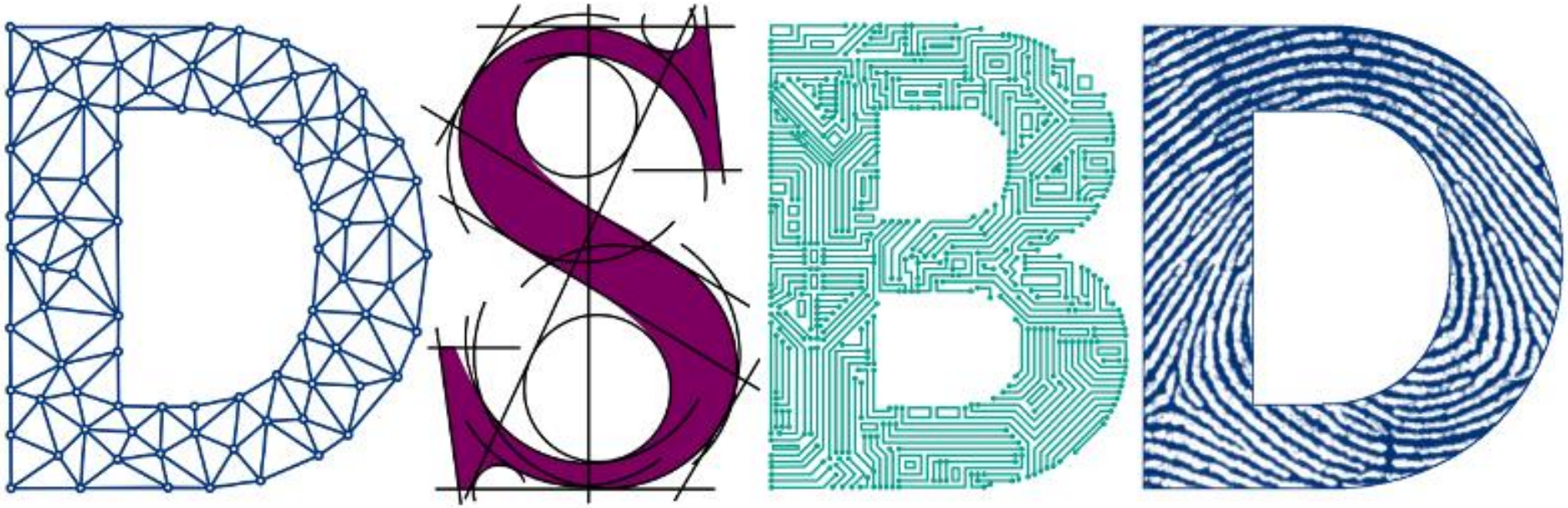


PROGRAMAÇÃO PARALELA FUNDAMENTOS E TERMINOLOGIAS

Infraesrutura Computacional Pt.3

Marco A. Z. Alves

PORQUÊ PROGRAMAÇÃO PARALELA?

"Se um único computador (processador) consegue resolver um problema em N segundos, podem N computadores (processadores) resolver o mesmo problema em 1 segundo?"

PORQUÊ PROGRAMAÇÃO PARALELA?

"Se um único computador (processador) consegue resolver um problema em N segundos, podem N computadores (processadores) resolver o mesmo problema em 1 segundo?"

"Podem N computadores (processadores) resolver o mesmo problema em menos de 1 segundo?"

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:

- Tirar proveito de recursos computacionais não disponíveis localmente ou subaproveitados.
- Ultrapassar limitações de memória quando a memória disponível num único computador é insuficiente para a resolução do problema.
- Ultrapassar os limites físicos de velocidade e de miniaturização que atualmente começam a restringir a possibilidade de construção de computadores sequenciais cada vez mais rápidos.

PORQUÊ PROGRAMAÇÃO PARALELA?

Tradicionalmente, a programação paralela foi motivada pela resolução/simulação de problemas fundamentais da ciência/engenharia de grande relevância científica e económica, denominados como Grand Challenge Problems (GCPs).

Tipicamente, os GCPs simulam fenômenos que não podem ser medidos por experimentação:

- Fenômenos climáticos (e.g. movimento das placas tectónicas)
- Fenômenos físicos (e.g. órbita dos planetas)
- Fenômenos químicos (e.g. reações nucleares)
- Fenômenos biológicos (e.g. genoma humano)
- Fenômenos geológicos (e.g. atividade sísmica)
- Componentes mecânicos (e.g. aerodinâmica/resistência de materiais em naves espaciais)
- Circuitos electrónicos (e.g. verificação de placas de computador)

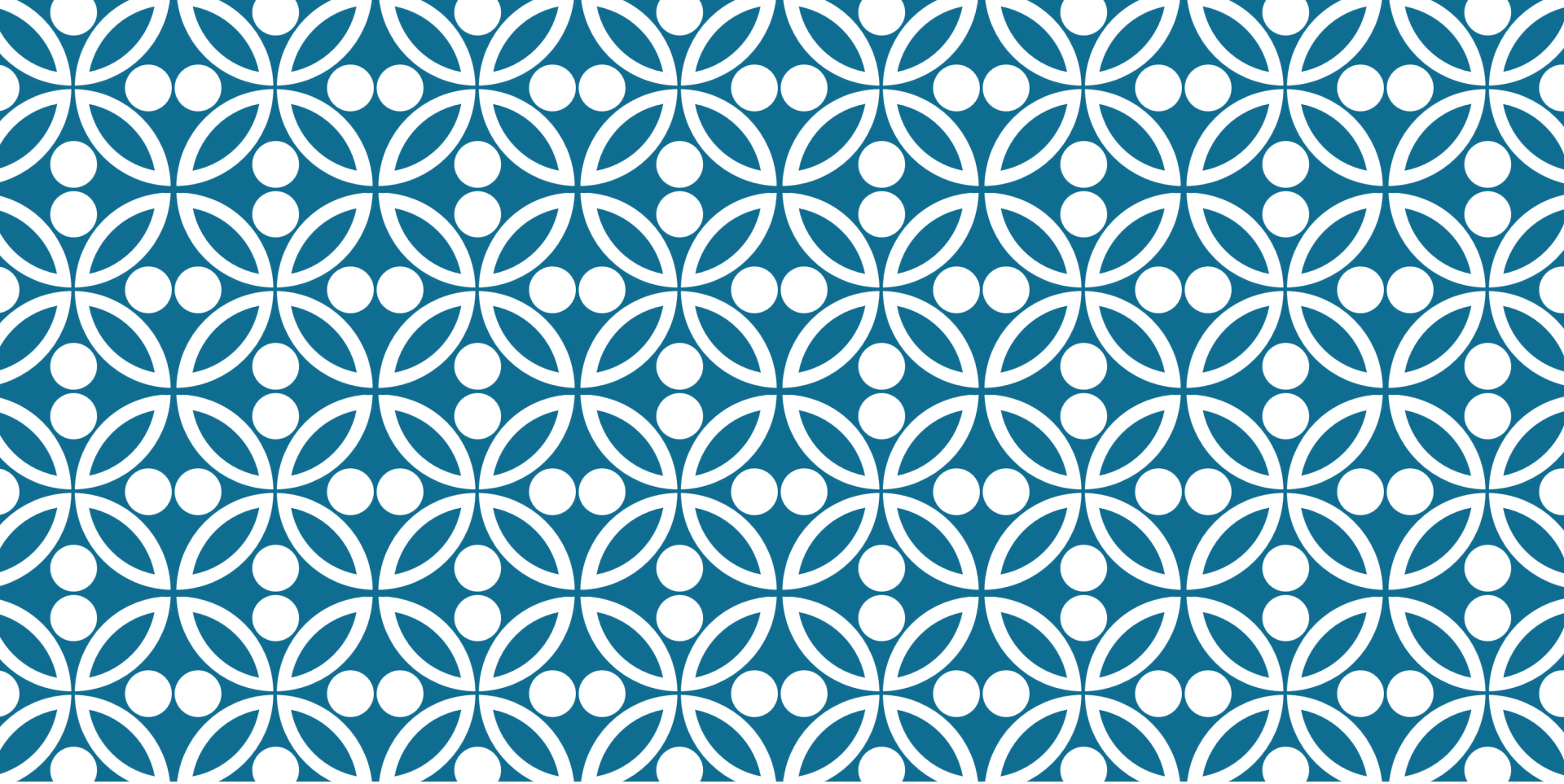
PORQUÊ PROGRAMAÇÃO PARALELA?

Atualmente, as aplicações que exigem o desenvolvimento de computadores cada vez mais rápidos estão por todo o lado.

Estas aplicações ou requerem um **grande poder de computação** ou requerem o processamento de **grandes quantidades de informação**.

Alguns exemplos são:

- Bases de dados paralelas
- Mineração de dados (data mining)
- Serviços de procura baseados na web
- Serviços associados a tecnologias multimídia e telecomunicações
- Computação gráfica e realidade virtual
- Diagnóstico médico assistido por computador
- Gestão de grandes industrias/corporações



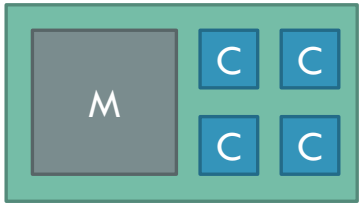
MODELOS DE MÁQUINAS PARALELAS

COMPUTAÇÃO PARALELA

De uma forma simples, a **computação paralela** pode ser definida como o uso simultâneo de vários recursos computacionais de forma a reduzir o tempo necessário para resolver um determinado problema.

Esses recursos computacionais podem incluir:

- **Um único computador com múltiplos processadores.**
- Um número arbitrário de computadores ligados por rede.
- A combinação de ambos.

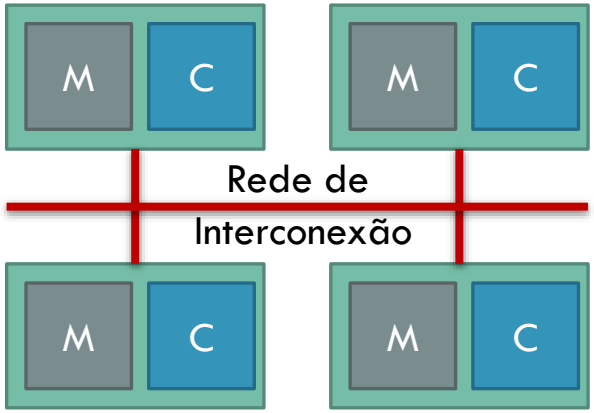


COMPUTAÇÃO PARALELA

De uma forma simples, a **computação paralela** pode ser definida como o uso simultâneo de vários recursos computacionais de forma a reduzir o tempo necessário para resolver um determinado problema.

Esses recursos computacionais podem incluir:

- Um único computador com múltiplos processadores.
- **Um número arbitrário de computadores ligados por rede.**
- A combinação de ambos.

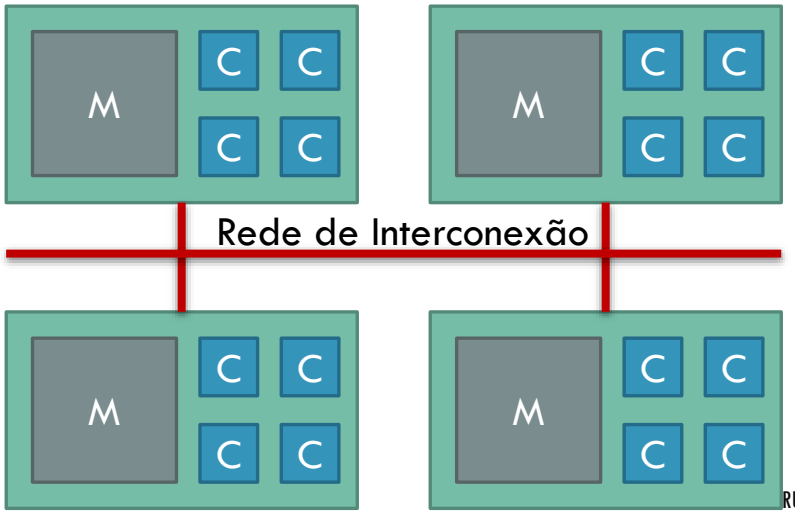


COMPUTAÇÃO PARALELA

De uma forma simples, a **computação paralela** pode ser definida como o uso simultâneo de vários recursos computacionais de forma a reduzir o tempo necessário para resolver um determinado problema.

Esses recursos computacionais podem incluir:

- Um único computador com múltiplos processadores.
- Um número arbitrário de computadores ligados por rede.
- **A combinação de ambos.**



MULTIPROCESSADORES VS. MULTICOMPUTADORES

Vantagens e inconvenientes: ???

- Partilha de dados
- Sincronização
- Escalabilidade
- Custo

MULTIPROCESSADORES VS. MULTICOMPUTADORES

Multi-Processadores

- (+) Partilha de dados entre tarefas é conseguida de forma simples, uniforme e rápida.
- (–) Necessita de mecanismos de sincronização para obter um correto manuseamento dos dados.
- (–) Pouco escalável. O aumento do número de processadores aumenta a contenção no acesso à memória e torna inviável qualquer mecanismo de coerência das caches.
- (–) Custo elevado. É difícil e bastante caro desenhar e produzir computadores cada vez com um maior número de processadores.

MULTIPROCESSADORES VS. MULTICOMPUTADORES

Multi-Processadores

- (+) Partilha de dados entre tarefas é conseguida de forma simples, uniforme e rápida.
- (–) Necessita de mecanismos de sincronização para obter um correto manuseamento dos dados.
- (–) Pouco escalável. O aumento do número de processadores aumenta a contenção no acesso à memória e torna inviável qualquer mecanismo de coerência das caches.
- (–) Custo elevado. É difícil e bastante caro desenhar e produzir computadores cada vez com um maior número de processadores.

Multi-Computadores

(+) O aumento do número de computadores aumenta proporcionalmente a memória disponível sem necessitar de mecanismos de coerência das caches.

(+) Fácil escalabilidade a baixo custo. O aumento do poder de computação pode ser conseguido à custa de computadores de uso doméstico.

(–) Necessita de mecanismos de comunicação para partilha de dados entre tarefas de diferentes computadores.

(–) O tempo de acesso aos dados entre diferentes computadores não é uniforme e é por natureza mais lento.

(–) Pode ser difícil converter estruturas de dados previamente existentes para memória partilhada em estruturas de dados para memória distribuída.

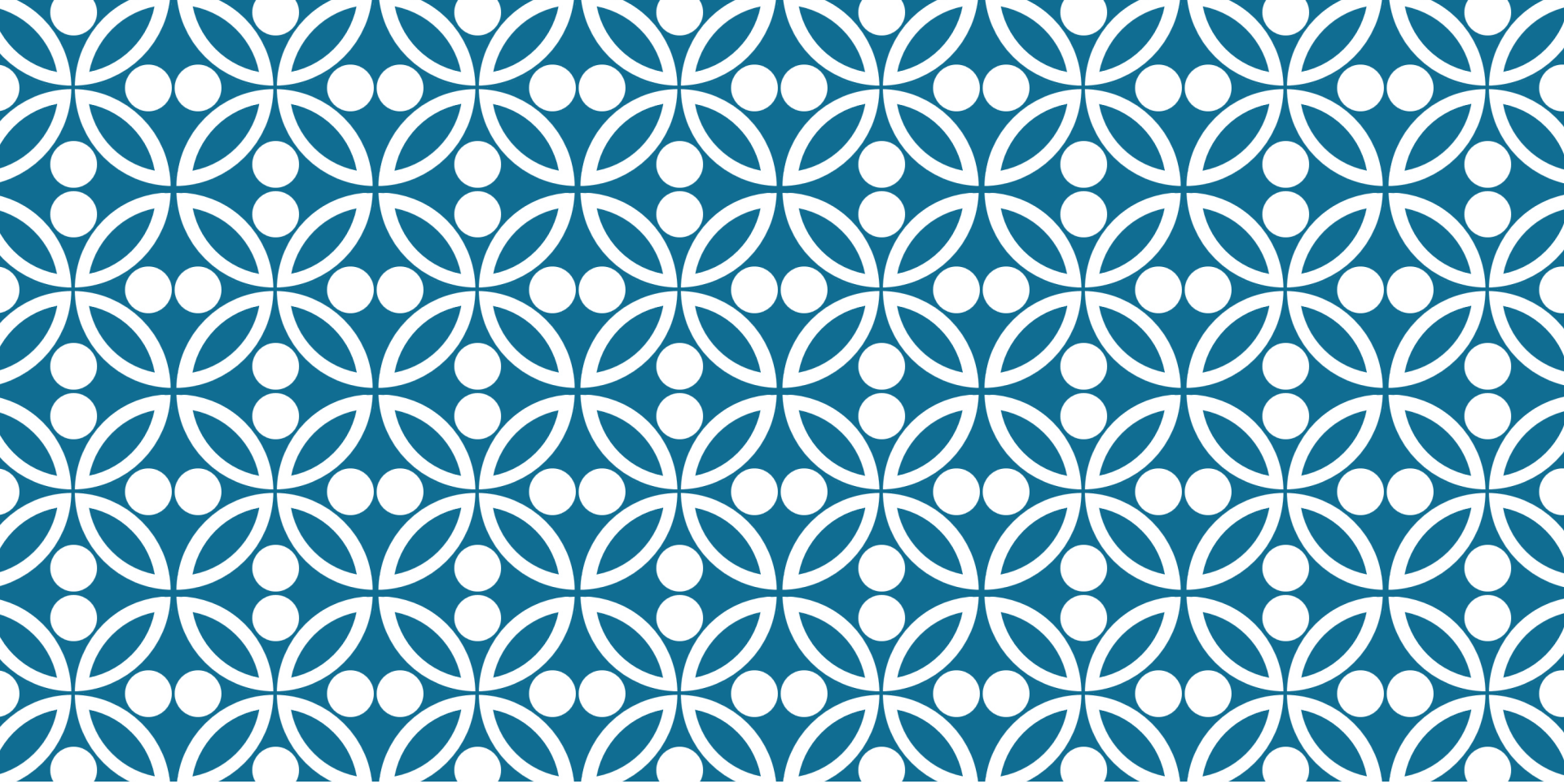
PRINCIPAIS MODELOS DE PROGRAMAÇÃO PARALELA

Programação em Memória Partilhada

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

Programação em Memória Distribuída

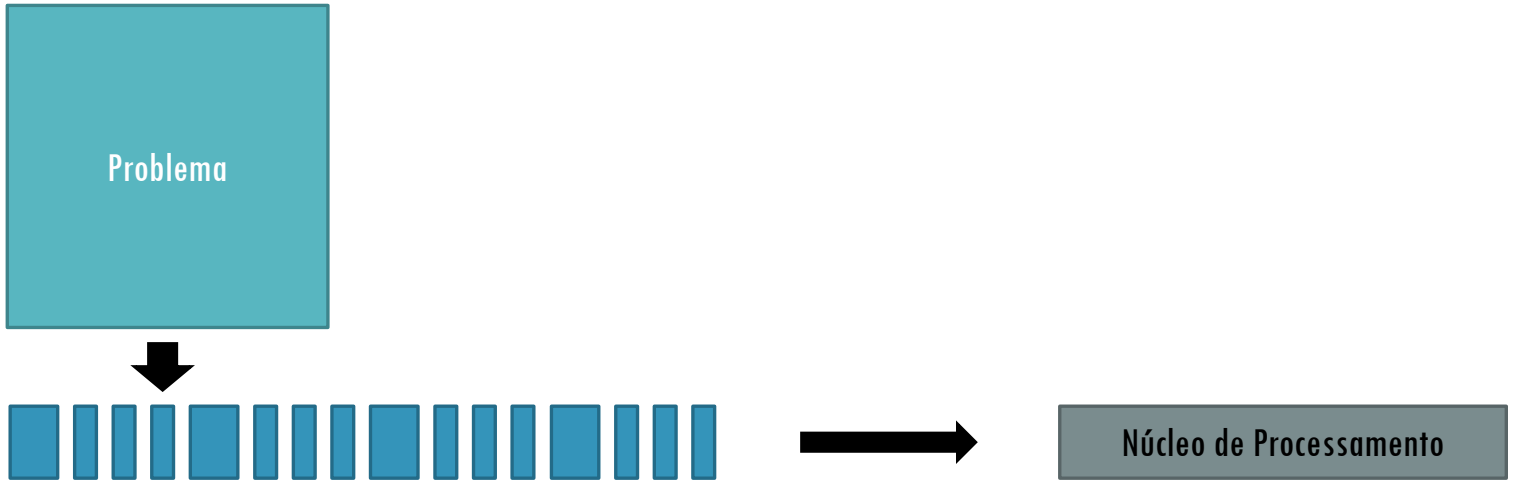
- Programação usando processos distribuídos
- Decomposição do domínio com granularidade grossa.
- Comunicação e sincronização por **troca de mensagens**.



PARALELISMO EXPLICITO

PROGRAMAÇÃO SEQUENCIAL

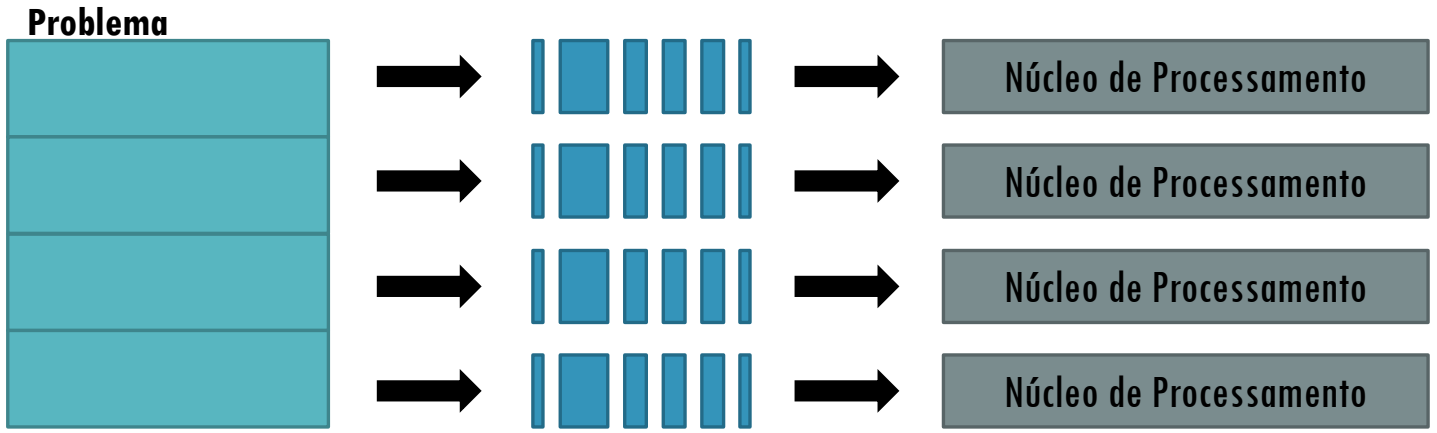
Considerado **programação sequencial** quando este é visto como uma série de instruções sequenciais que devem ser executadas num único processador.



PROGRAMAÇÃO PARALELA

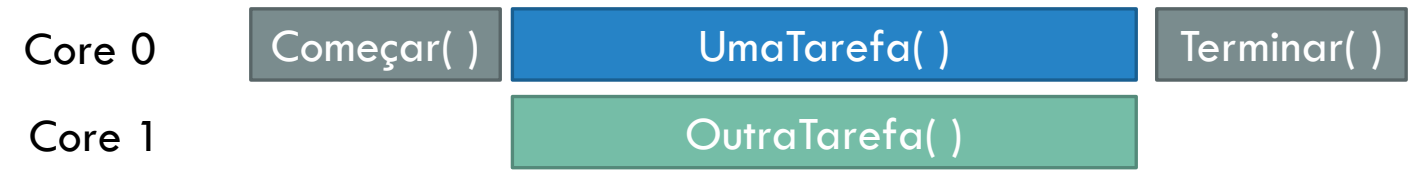
Considerado **programação paralela** quando este é visto como um conjunto de partes que podem ser resolvidas concorrentemente.

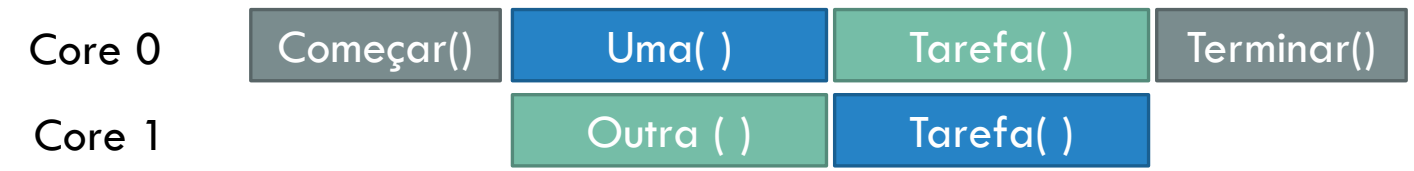
Cada parte é igualmente constituída por uma série de instruções sequenciais, mas que no seu conjunto podem ser executadas simultaneamente.



PARALELISMO

Paralelismo diz-se quando as tarefas de um programa são executadas em simultâneo em mais do que um processador / núcleo de processamento





Mesmo que ocorram **migrações**!

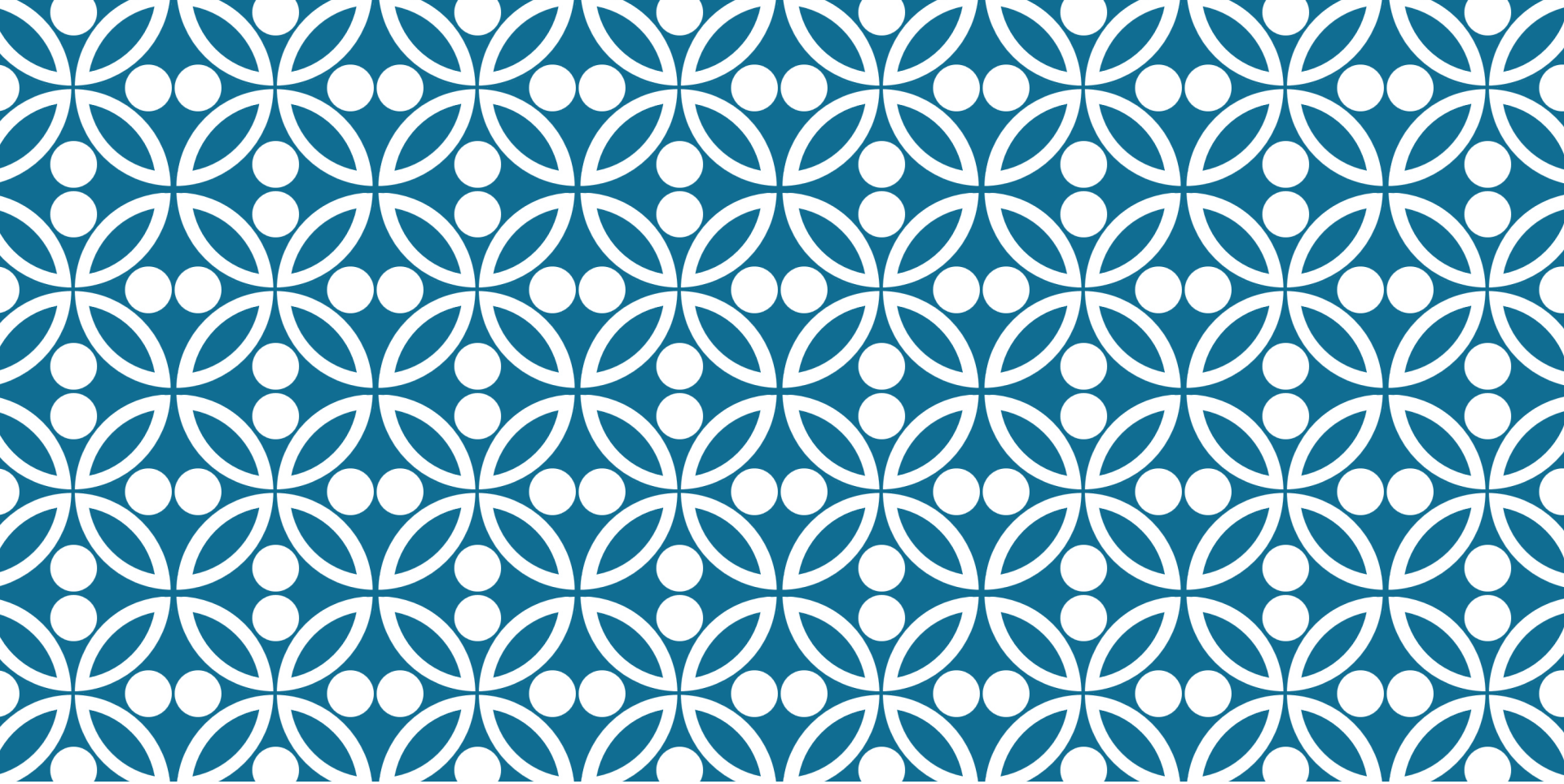
EXPLICITAR O PARALELISMO

Anotar as tarefas para execução em paralelo.

Atribuir (possivelmente) as tarefas aos processadores.

Controlar a execução indicando os pontos de sincronização.

Conhecer a arquitetura dos computadores de forma a conseguir o máximo desempenho (aumentar localidade, diminuir comunicação, etc).



MIGRAÇÃO PROGRAMAS SEQUENCIAL→ PARALELO

PROGRAMAÇÃO PARALELA

Apesar das arquiteturas paralelas serem atualmente uma realidade, a **programação paralela continua a ser uma tarefa complexa.**

Debate-se com uma série de **problemas não existentes** em programação sequencial:

PROGRAMAÇÃO PARALELA

Apesar das arquiteturas paralelas serem atualmente uma realidade, a programação paralela continua a ser uma tarefa complexa.

Debate-se com uma série de problemas não existentes em programação sequencial:

Concorrência: identificar as partes da computação que podem ser executadas em simultâneo.

PROGRAMAÇÃO PARALELA

Apesar das arquiteturas paralelas serem atualmente uma realidade, a programação paralela continua a ser uma tarefa complexa.

Debate-se com uma série de problemas não existentes em programação sequencial:

Concorrência: identificar as partes da computação que podem ser executadas em simultâneo.

Comunicação e Sincronização: desenhar o fluxo de informação de modo a que a computação possa ser executada em simultâneo pelos diversos processadores evitando situações de deadlock e race conditions.

PROGRAMAÇÃO PARALELA

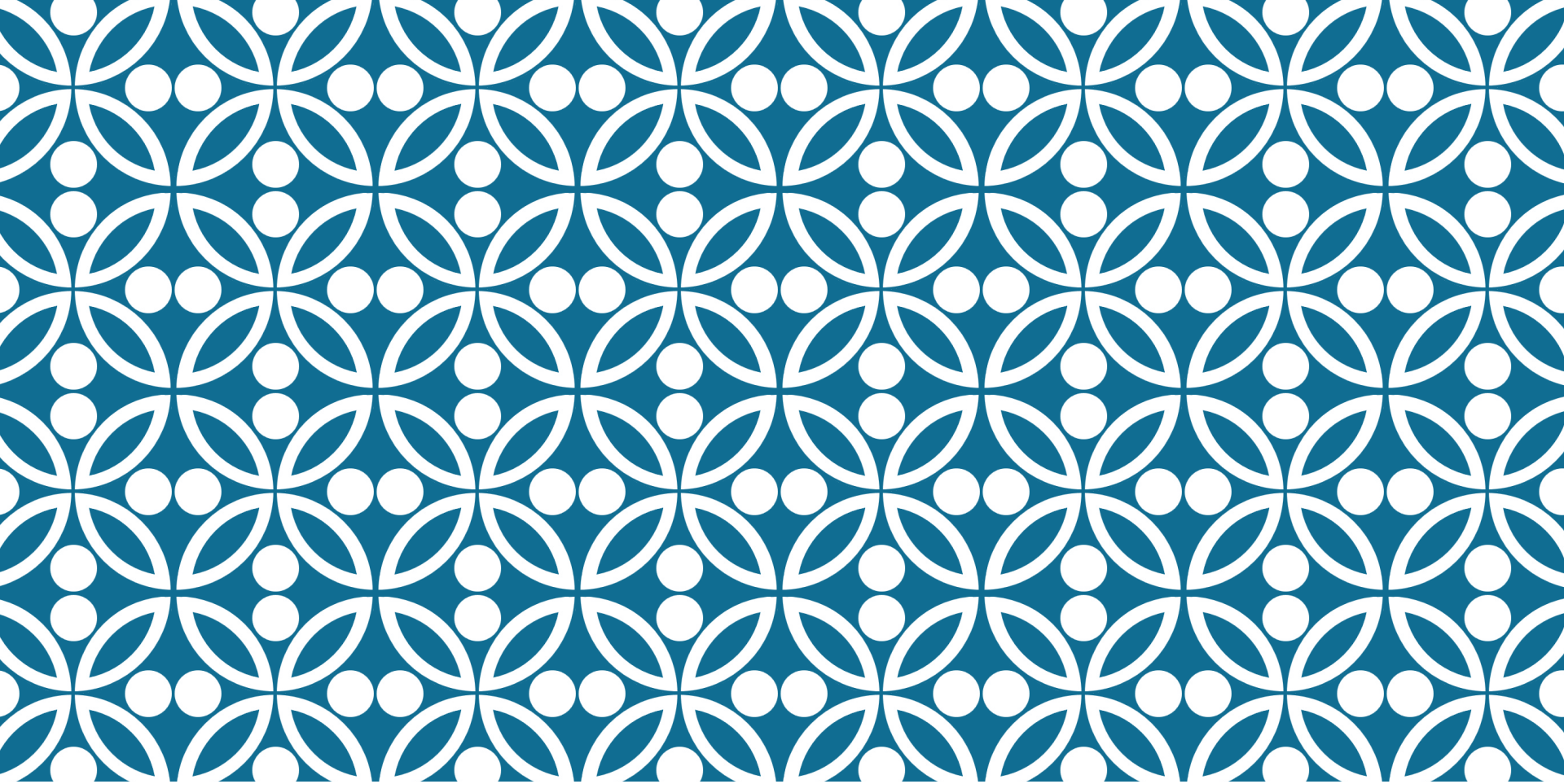
Apesar das arquiteturas paralelas serem atualmente uma realidade, a programação paralela continua a ser uma tarefa complexa.

Debate-se com uma série de problemas não existentes em programação sequencial:

Concorrência: identificar as partes da computação que podem ser executadas em simultâneo.

Comunicação e Sincronização: desenhar o fluxo de informação de modo a que a computação possa ser executada em simultâneo pelos diversos processadores evitando situações de deadlock e race conditions.

Balanceamento de Carga e Escalonamento: distribuir de forma equilibrada e eficiente as diferentes partes da computação pelos diversos processadores de modo a ter os processadores maioritariamente ocupados durante toda a execução.



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).

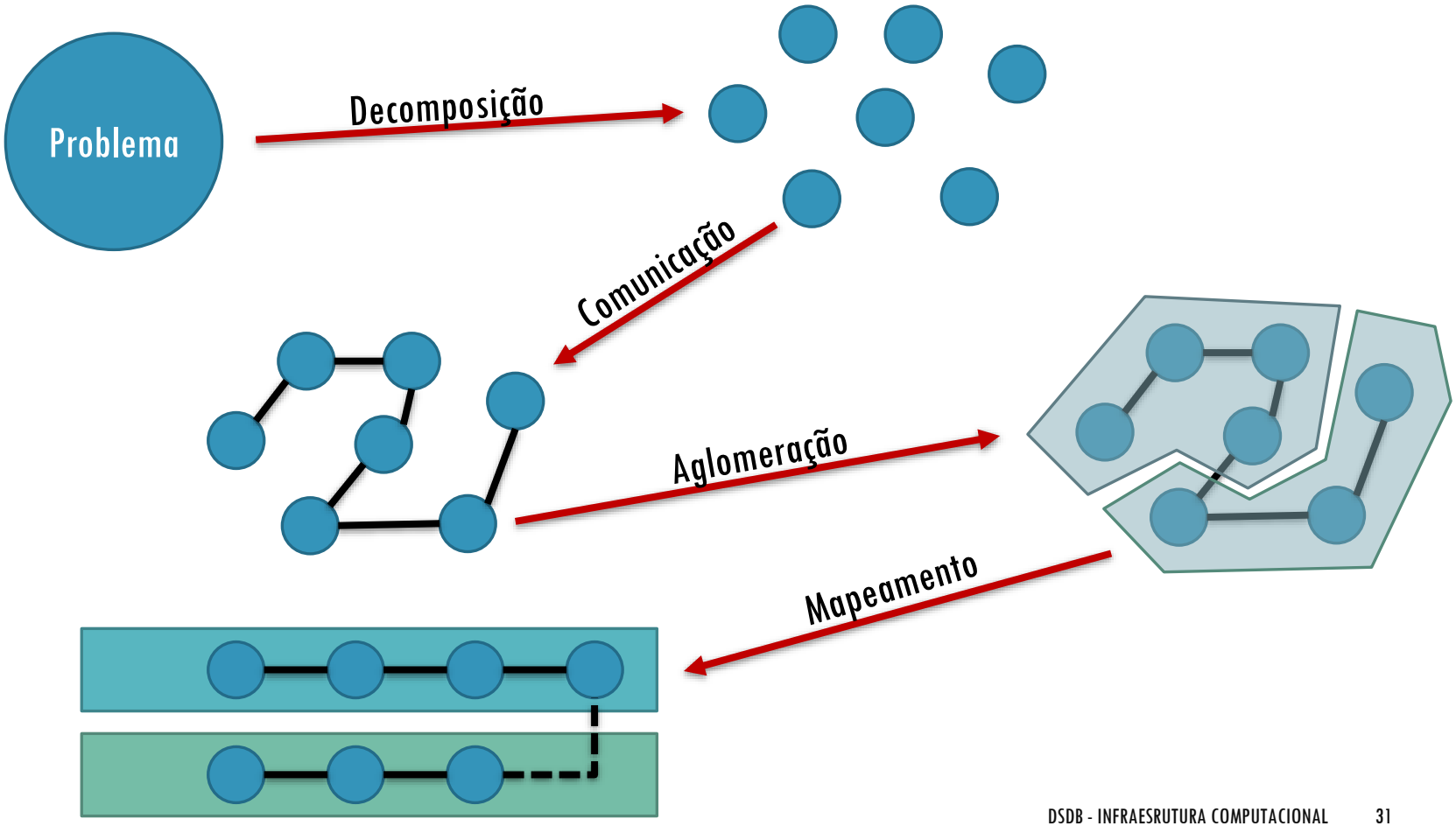
Esta metodologia permite que o programador se concentre inicialmente nos aspectos não-dependentes da arquitetura, como sejam a concorrência e a escalabilidade

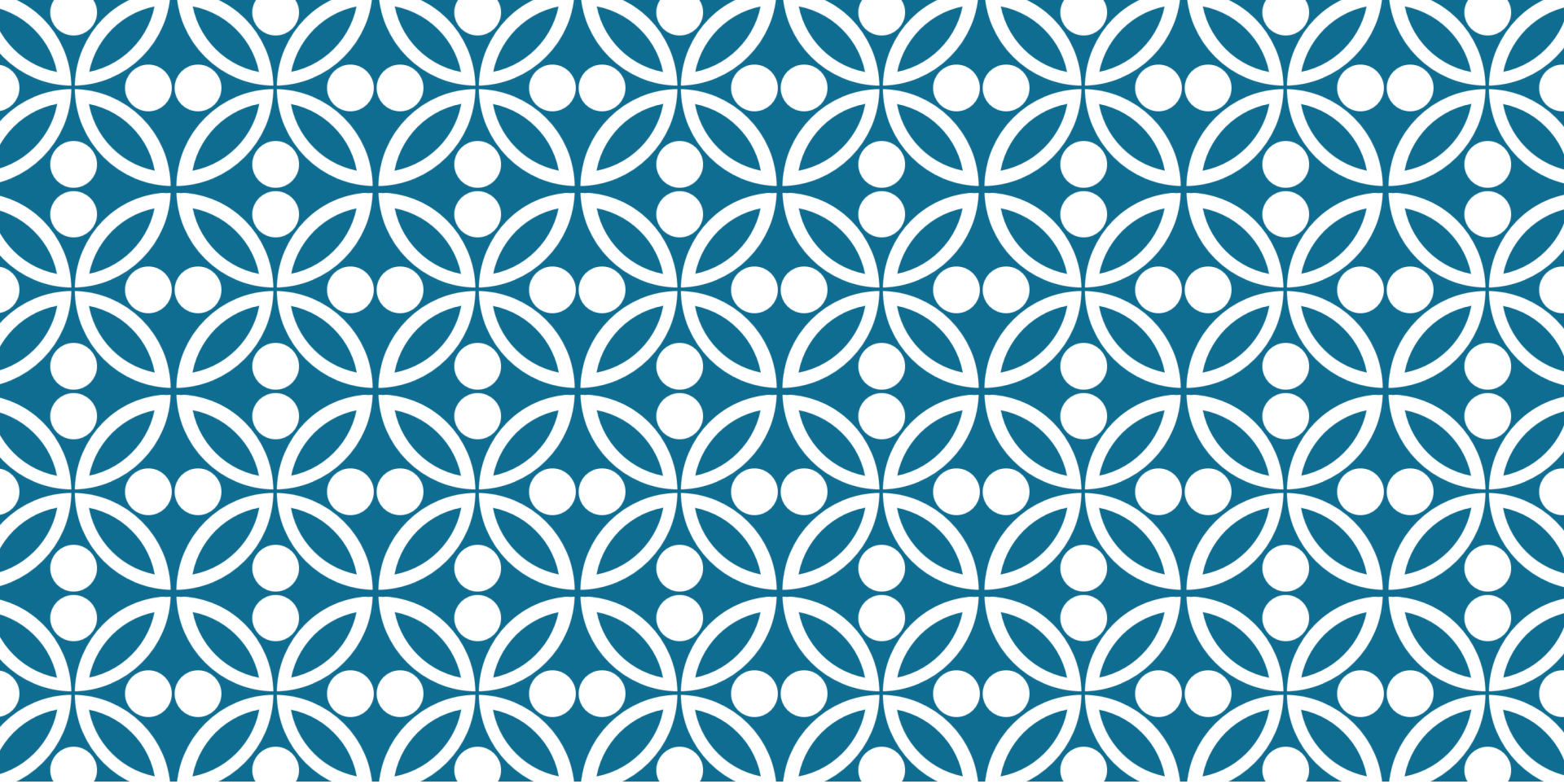
Somente depois deve-se considerar os aspectos dependentes da arquitetura, como sejam aumentar a localidade e diminuir a comunicação da computação.

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

- Decomposição
- Comunicação
- Aglomeração
- Mapeamento

METODOLOGIA DE PROGRAMAÇÃO DE FOSTER





FOSTER – ETAPA 1 DECOMPOSIÇÃO

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:

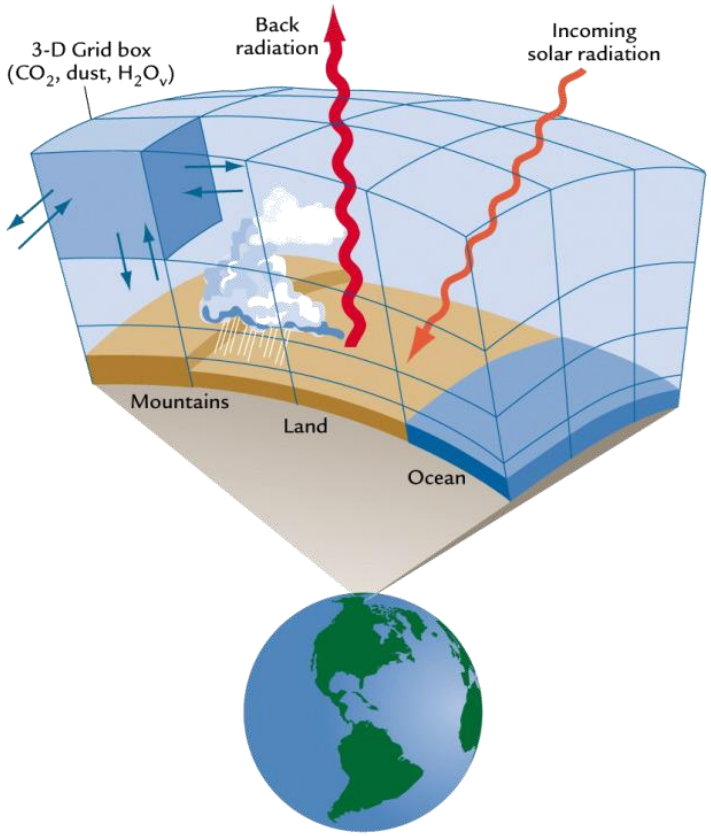
- Decomposição do Domínio: decompor o problema em função dos dados.
- Decomposição Funcional: decompor o problema em função da computação.

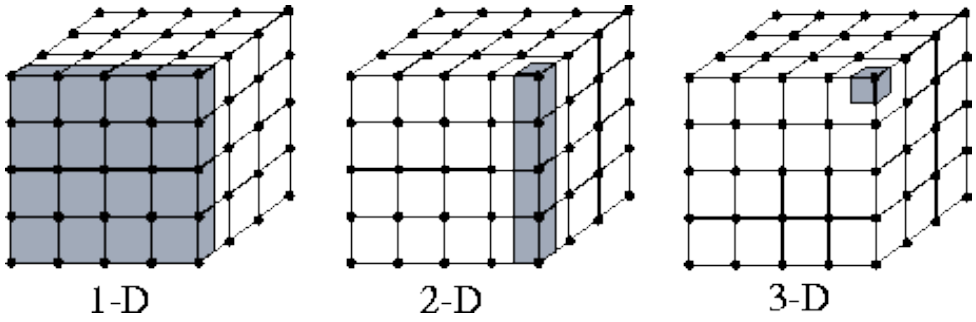
Um boa decomposição tanto divide os dados como a computação em múltiplas tarefas mais pequenas.

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.



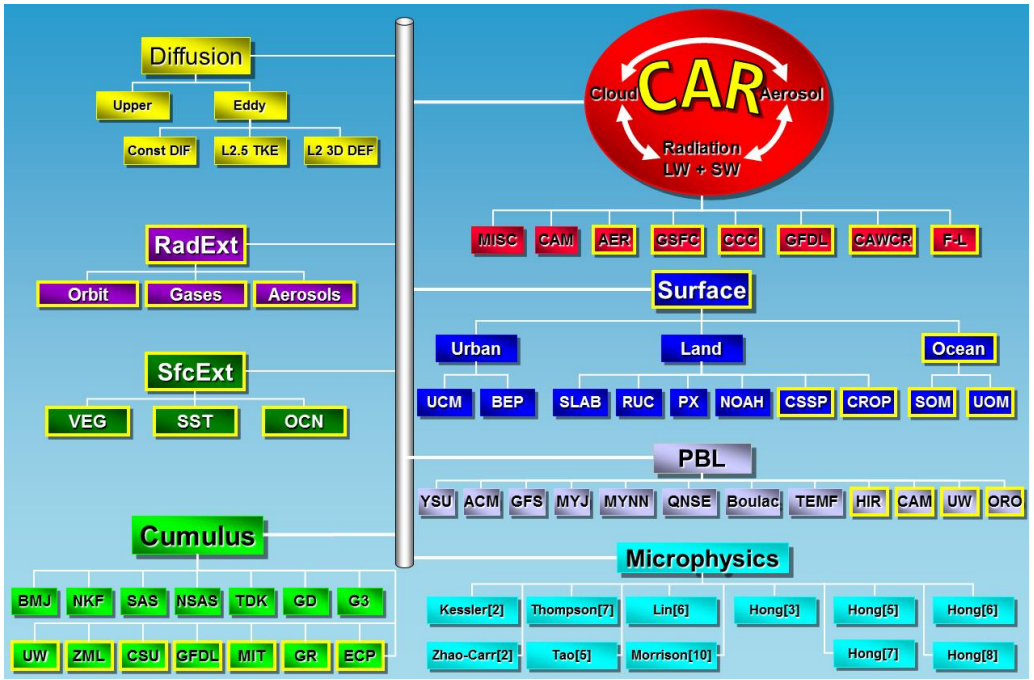


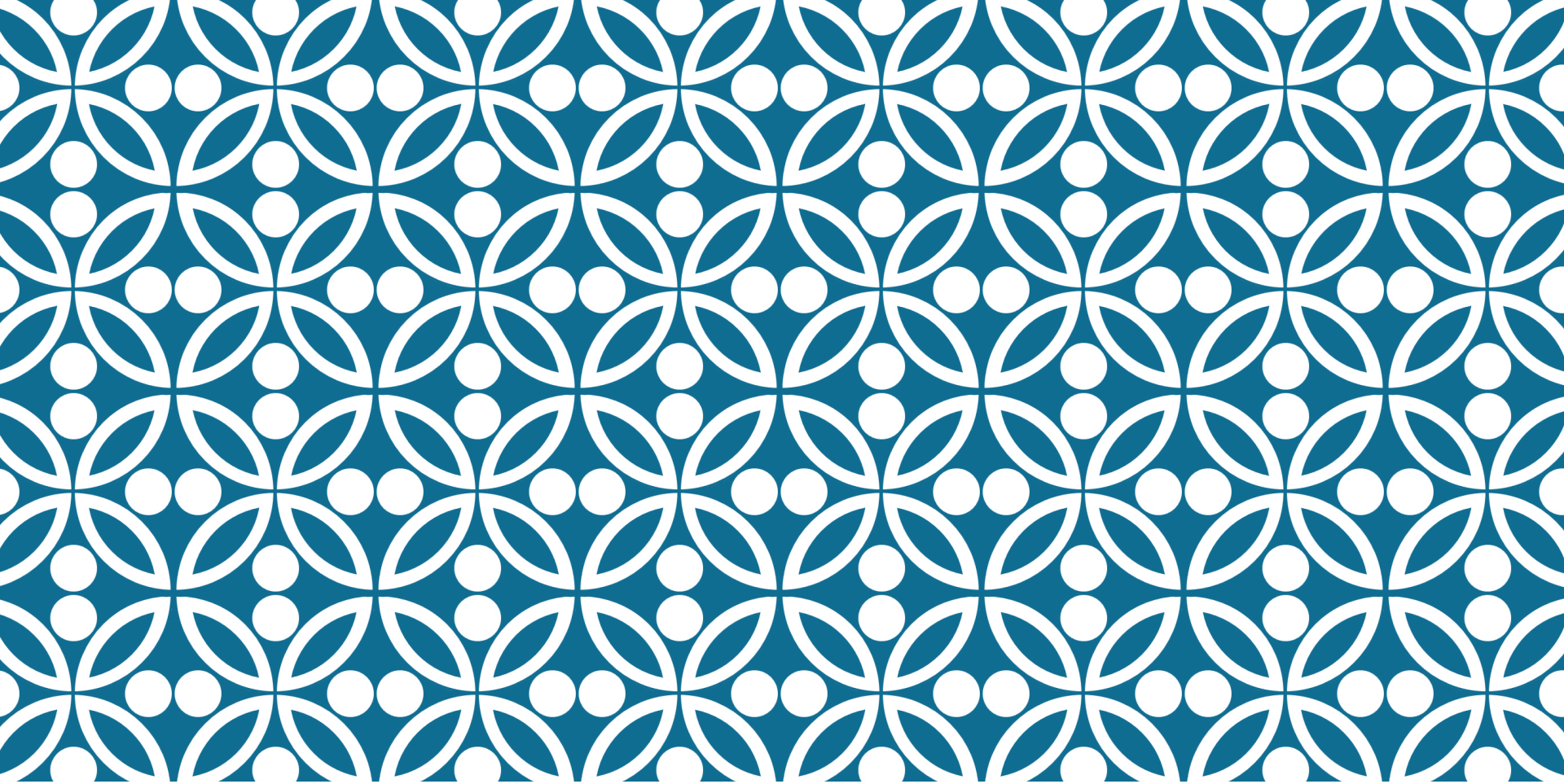
DECOMPOSIÇÃO FUNCIONAL

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.





FOSTER – ETAPA 2 COMUNICAÇÃO

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:

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

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:

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

Necessidade de Sincronização: enquanto as tarefas ficam à espera de sincronizar não contribuem para a computação.

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:

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

Necessidade de Sincronização: enquanto as tarefas ficam à espera de sincronizar não contribuem para a computação.

Latência (tempo mínimo de comunicação entre dois pontos) e **Largura de Banda** (quantidade de informação comunicada por unidade de tempo): é boa prática enviar poucas mensagens grandes do que muitas mensagens pequenas.

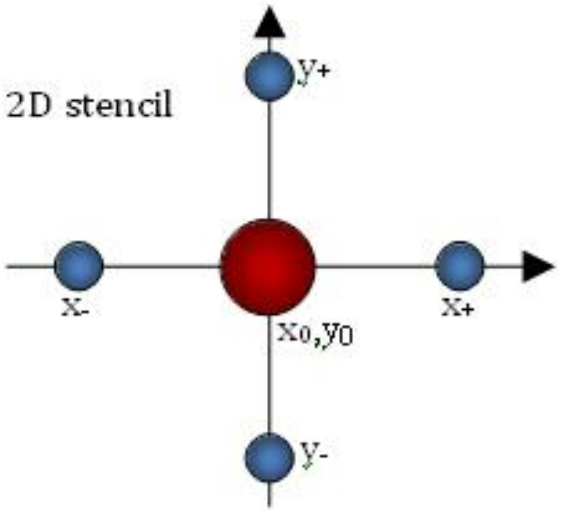
PADRÕES DE COMUNICAÇÃO

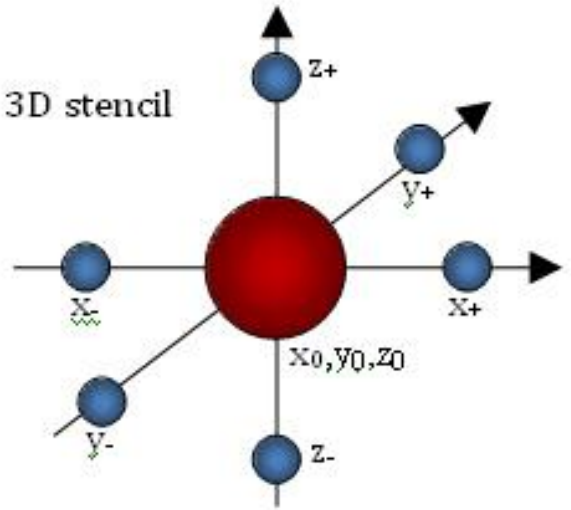
Comunicação Global

- Todas as tarefas podem comunicar entre si.

Comunicação Local

- A comunicação é restrita a tarefas vizinhas (e.g. método de Jacobi de diferenças finitas).





PADRÕES DE COMUNICAÇÃO

Comunicação Estruturada

- Tarefas vizinhas constituem uma estrutura regular (e.g. árvore ou rede).

Comunicação Não-Estruturada

- Comunicação entre tarefas constituí um grafo arbitrário.

PADRÕES DE COMUNICAÇÃO

Comunicação Estruturada

- Tarefas vizinhas constituem uma estrutura regular (e.g. árvore ou rede).

Comunicação Não-Estruturada

- Comunicação entre tarefas constituí um grafo arbitrário.

Comunicação Estática

- Os parceiros de comunicação não variam durante toda a execução.

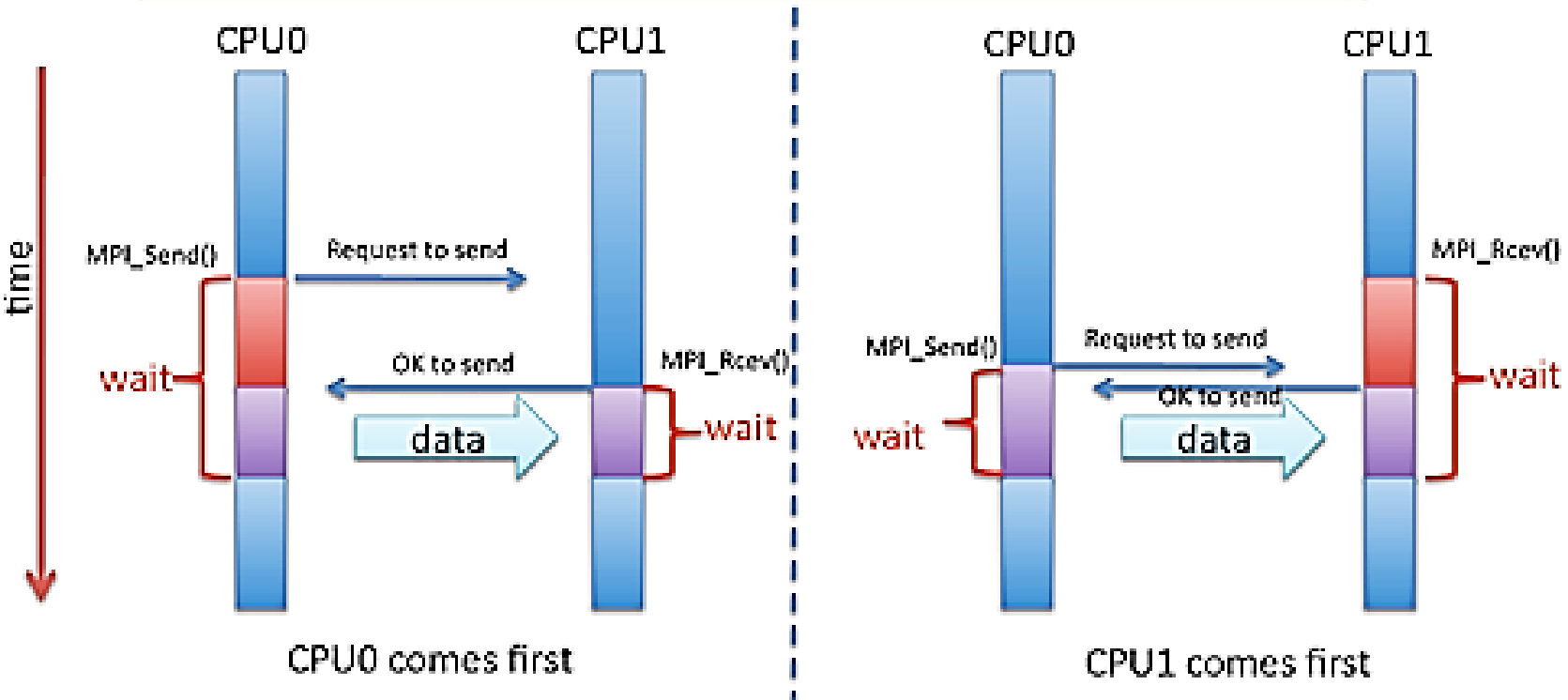
Comunicação Dinâmica

- A comunicação é determinada pela execução e pode ser muito variável.

PADRÕES DE COMUNICAÇÃO

Comunicação Bloqueante

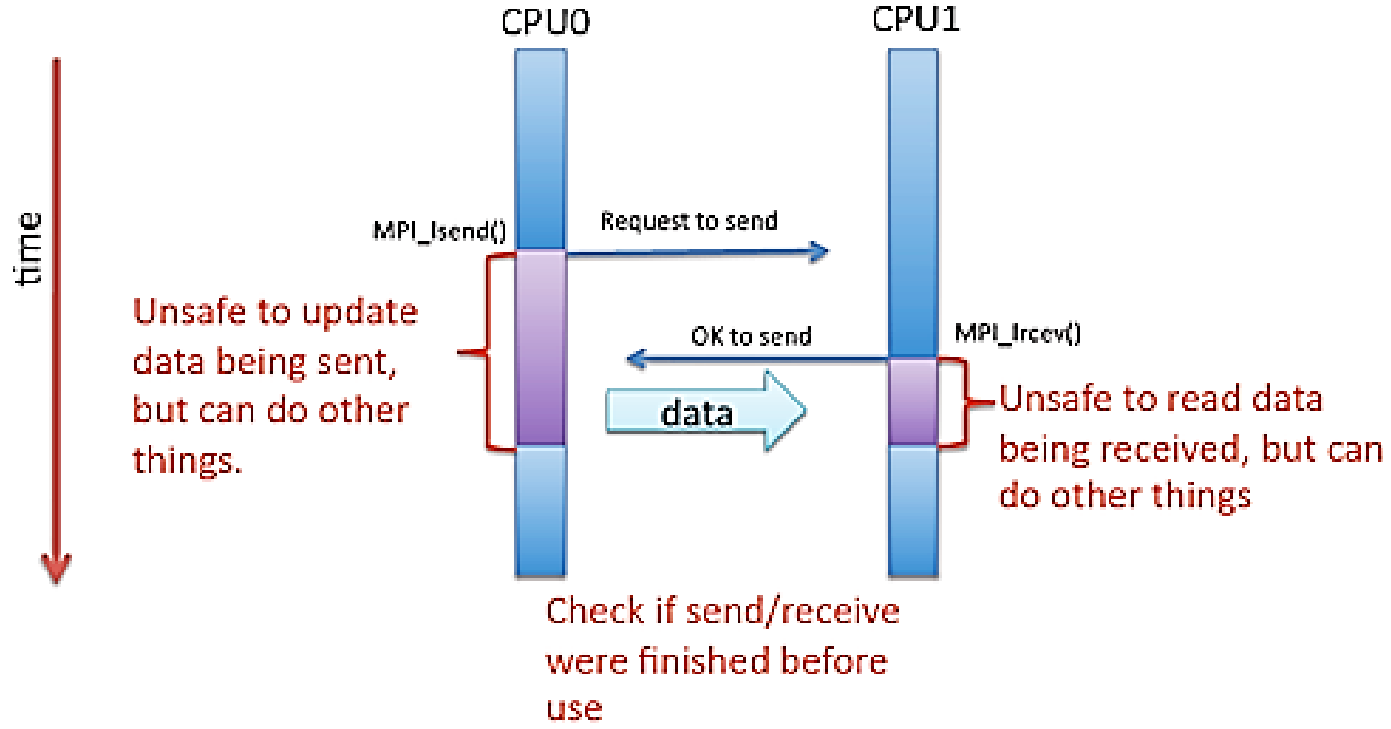
- As tarefas executam de forma coordenada e sincronizam na transferência de dados (e.g. protocolo das 3-fases ou rendez-vous: a comunicação apenas se concretiza quando as duas tarefas estão sincronizadas).

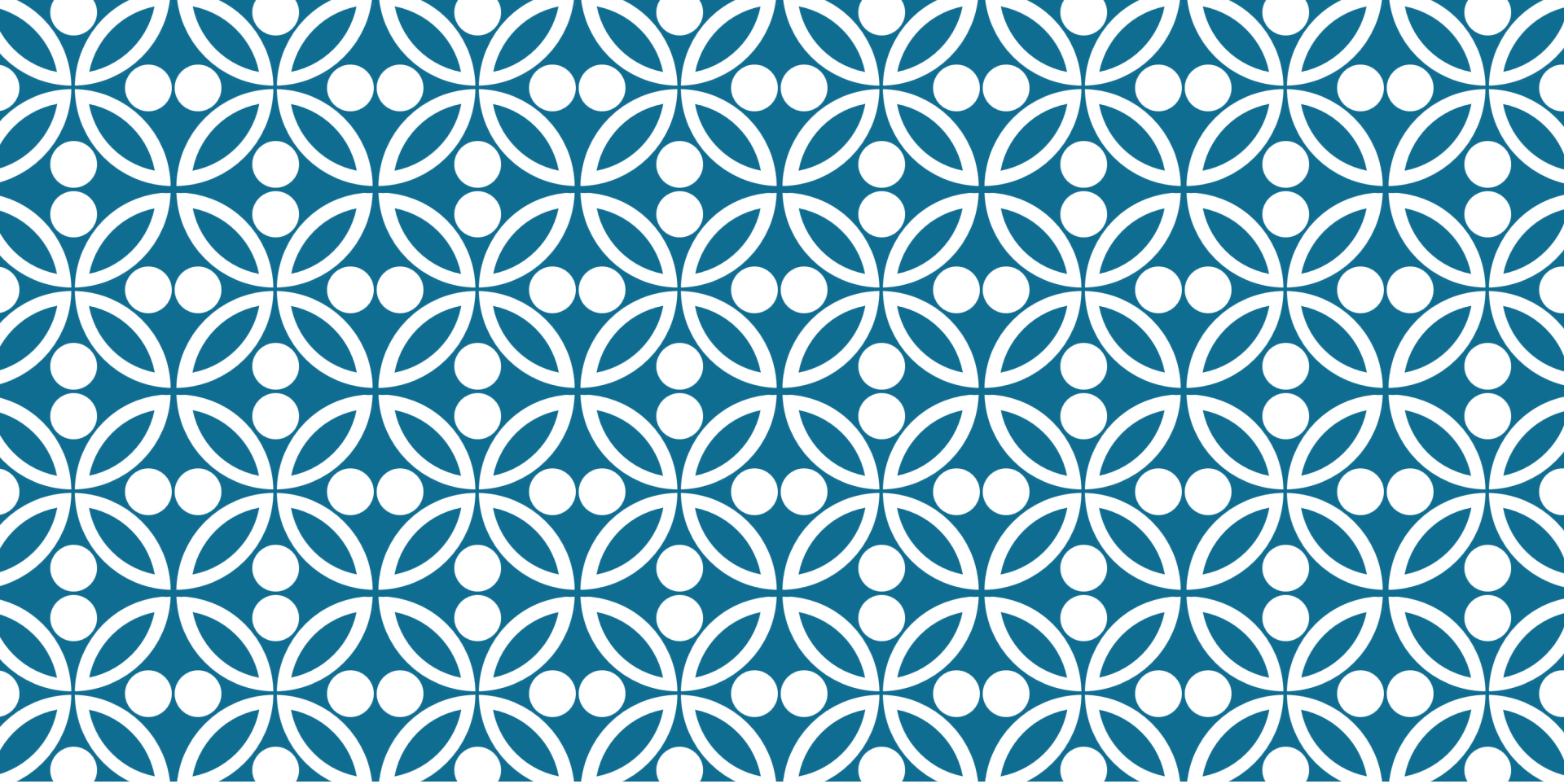


PADRÕES DE COMUNICAÇÃO

Comunicação Não-Bloqueante

- As tarefas executam de forma independente não necessitando de sincronizar para transferir dados (e.g. buffering de mensagens: o envio de mensagens não interfere com a execução do emissor).





FOSTER – ETAPA 3 AGLOMERAÇÃO

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).

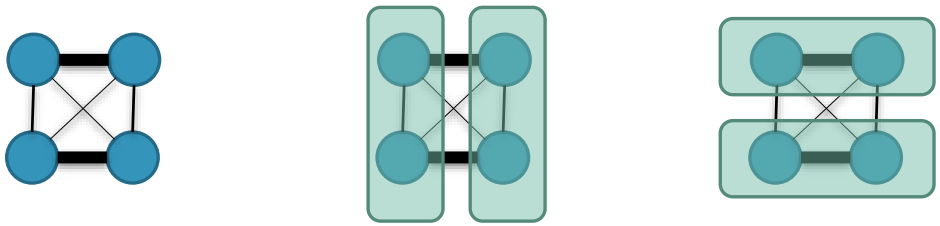
AGLOMERAÇÃO

A comunicação pode ser considerada em termos de **mensagens entre processos** ou em termos de **memória compartilhada entre threads**

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

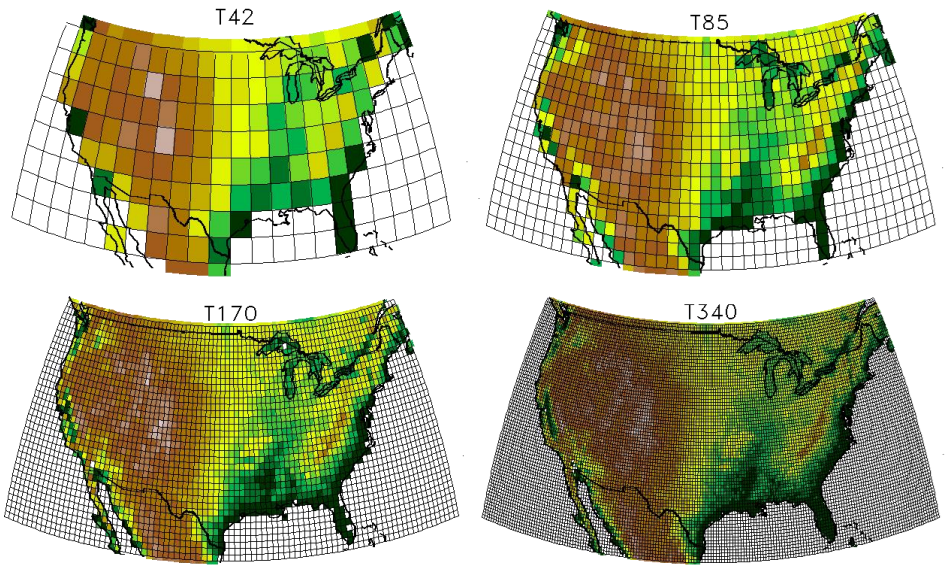
“Como agrupar a computação de modo a obter o máximo desempenho?”

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**.



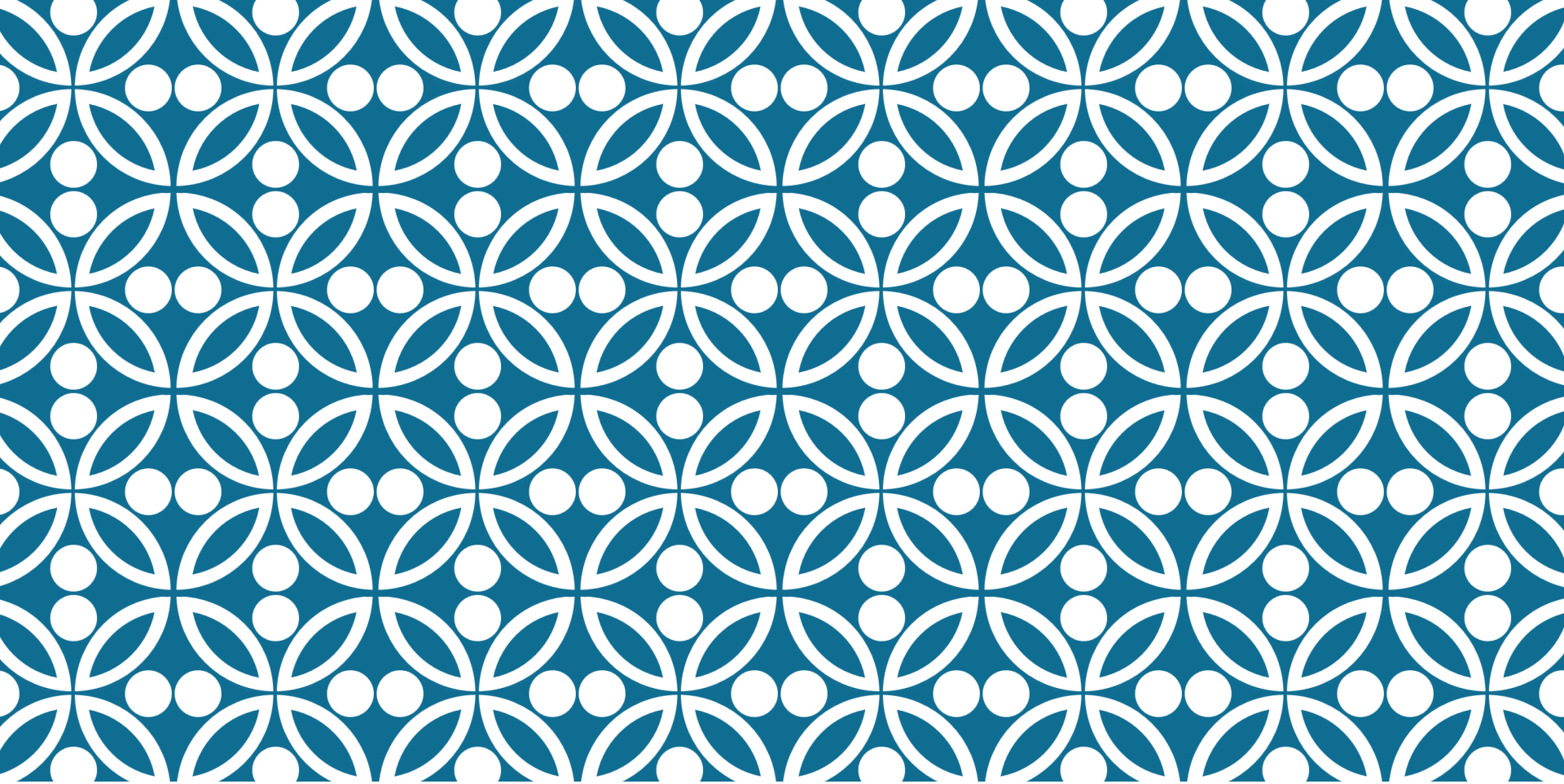
GRANULARIDADE

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.



FOSTER – ETAPA 4 MAPEAMENTO

MAPEAMENTO

"Como conseguir o melhor compromisso entre maximizar ocupação e minimizar comunicação?"

Mapeamento é o processo de atribuir tarefas a processadores de modo a maximizar a percentagem de ocupação e minimizar a comunicação entre processadores.

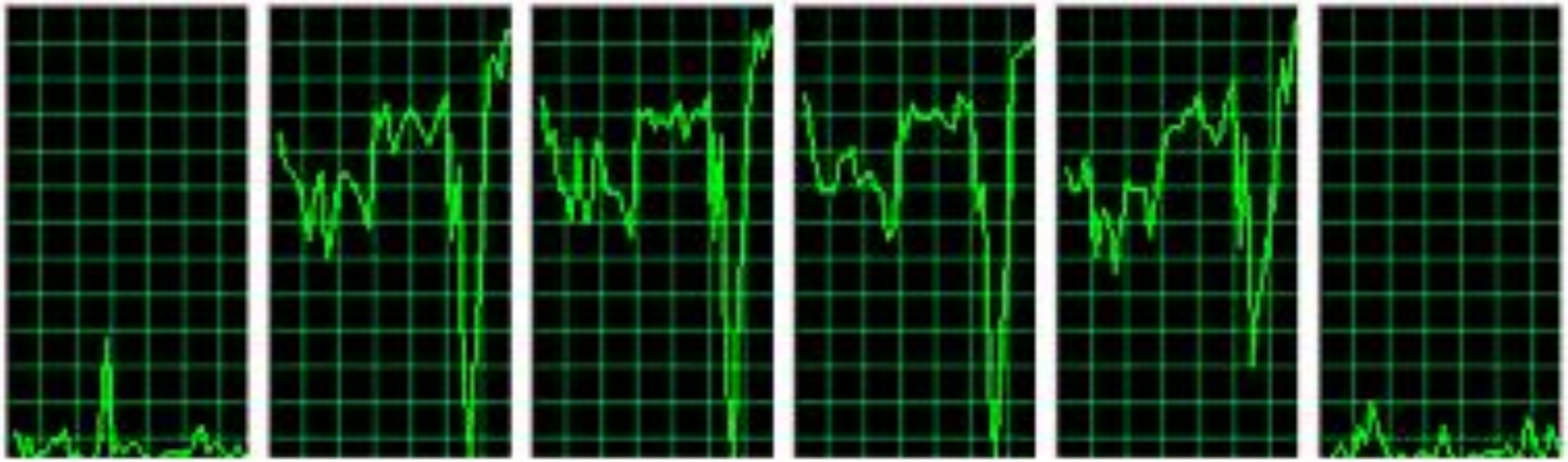
A percentagem de **ocupação é ótima** quando a computação é balanceada de forma igual pelos processadores, permitindo que todos começam e terminem as suas tarefas em simultâneo.

A percentagem de **ocupação decresce** quando um ou mais processadores ficam suspensos enquanto os restantes continuam ocupados.

A comunicação entre processadores é menor quando tarefas que comunicam entre si são atribuídas ao mesmo processador. No entanto, este mapeamento nem sempre é compatível com o objetivo de maximizar a percentagem de ocupação.

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.

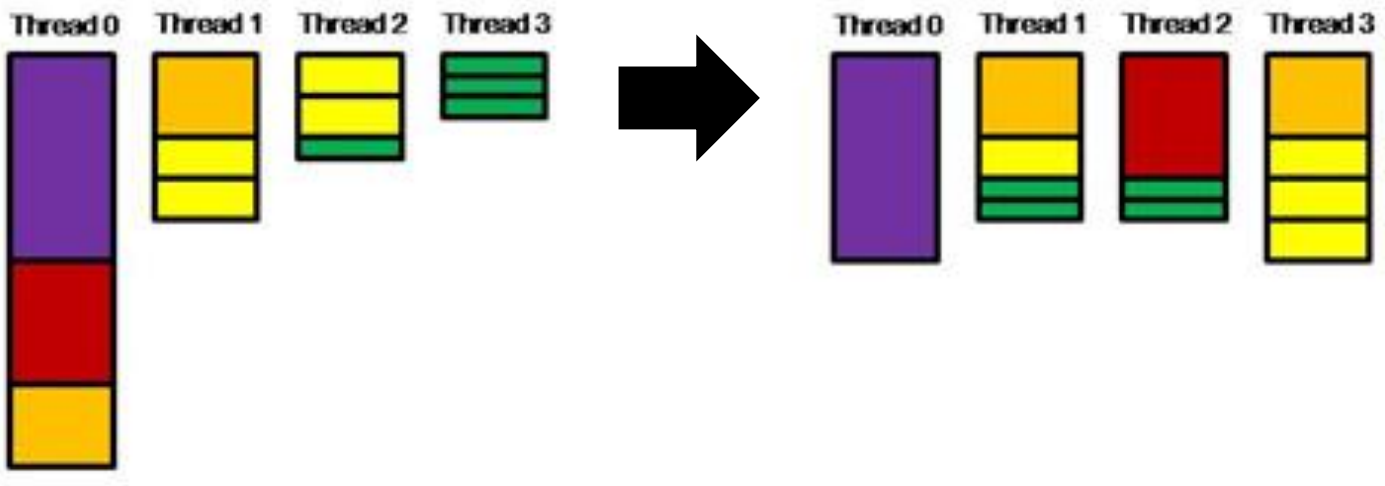


BALANCEAMENTO DE CARGA

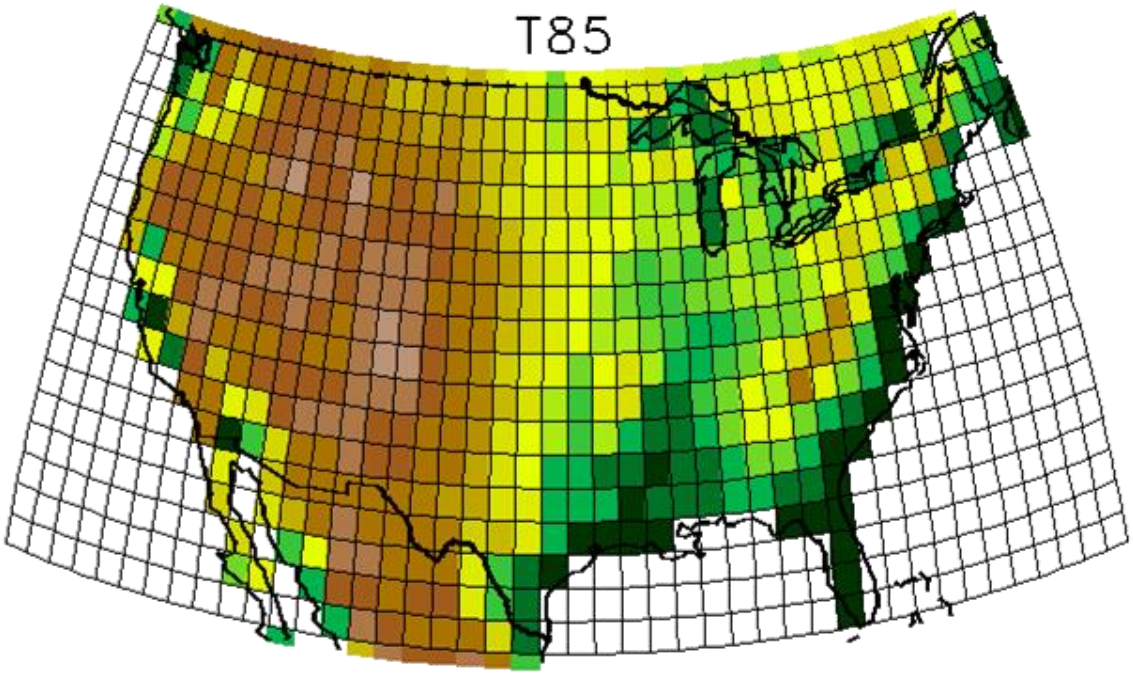
O balanceamento de carga refere-se à capacidade de distribuir tarefas pelos processadores de modo a que todos os processadores estejam ocupados todo o tempo.

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).

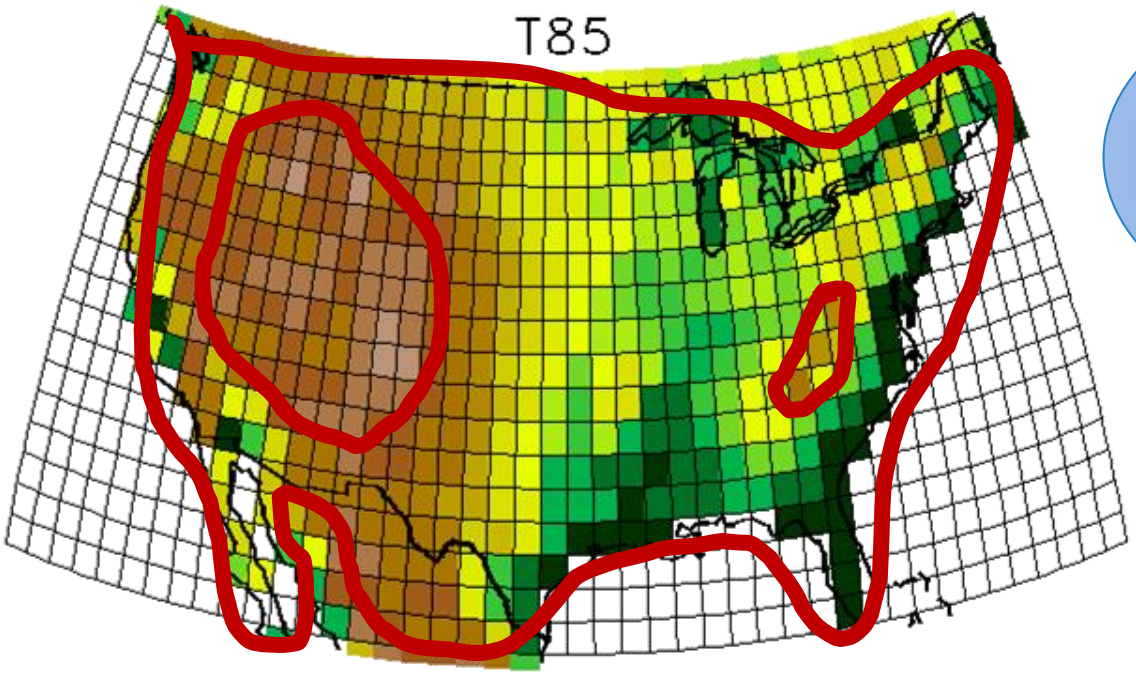


BALANCEAMENTO DE CARGA



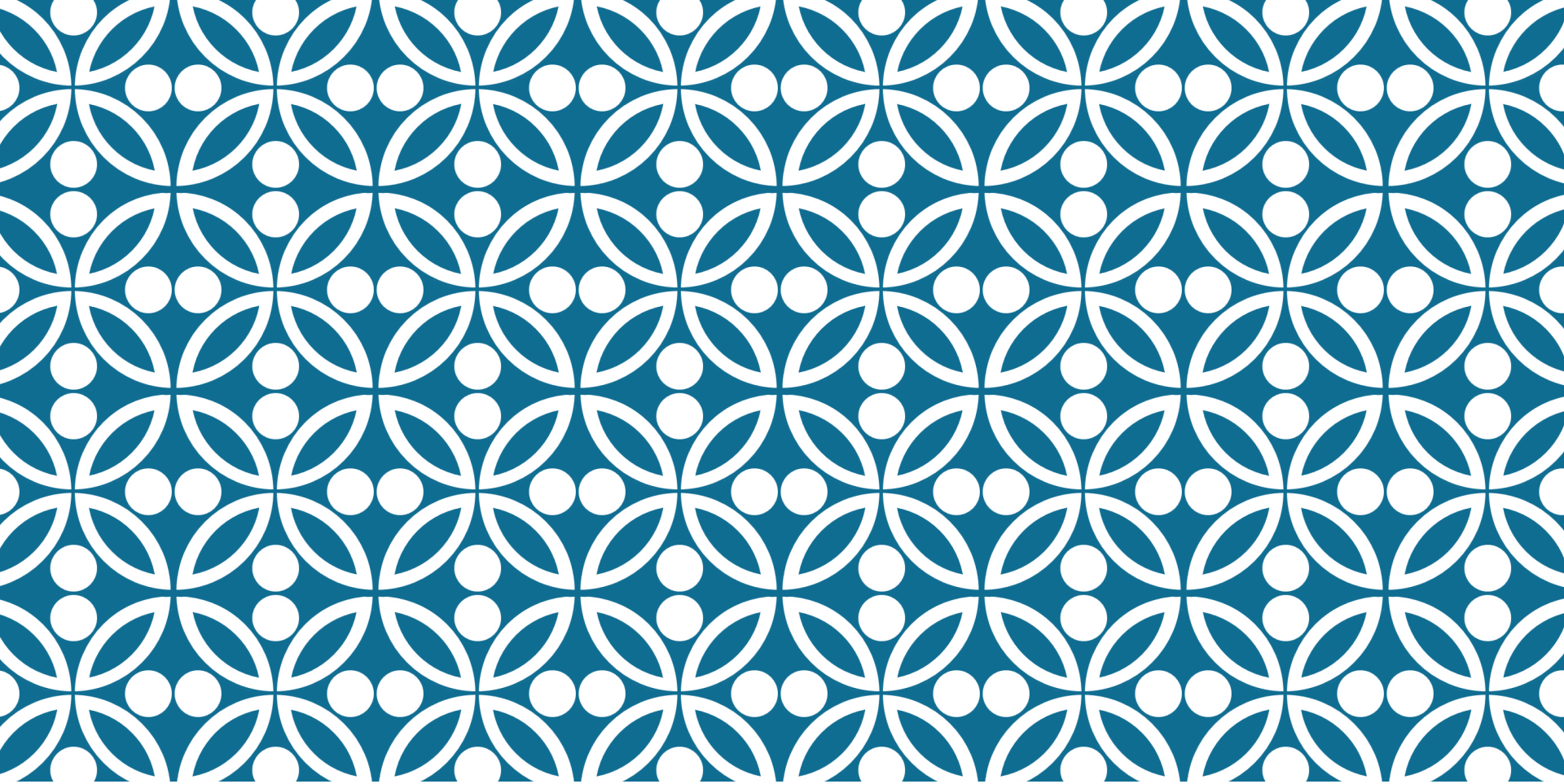
Como dividir esse problema para evitar desbalanceamento de carga?

BALANCEAMENTO DE CARGA



Pode depender das propriedades do problema a ser resolvido!

Continente ou Sol ou Calor ➔ muito trabalho de simulação



RESUMO

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.