Padrões de Programação Paralela

Serialização, Fork-Join,
Map, Reduction, Stencil etc.

Programação Estruturada

A “programação estruturada” (sequencial) é composta pelos seguintes **padrões**:

- Estruturas de Seleção e Repetição
- Funções, Recursão etc.

Estas **boas práticas** de programação eliminam problemas como o GOTO

- Melhora a manutenabilidade do SW.
- Aumenta o reuso de SW.
- Problema: Serialização

Serialização

Abstração de uma máquina sequencial.

- Possui ordem temporal.
- É determinística
  - Fácil de codificar, depurar e testar.

Limita o paralelismo sem necessidade.

- Paralelismo não é determinístico.

Exemplo Serialização

Procurar se uma palavra aparece ou não em cada página de um conjunto de páginas Web.

Como ler este código?

- Buscar “palavra” na página i, depois em i+1, depois em i+2 etc.

```
for (i = 0; i < numPaginas; i++) {
     encontrada[i] = busca(“palavra”,paginas[i]);
}
```

Exemplo Paralelização

A busca na página[i] precisa acontecer antes da busca na página[i+1]?

- As operações de busca são completamente independentes, ou seja, podem ser realizadas ao mesmo tempo.

```
parallel_for (i = 0; i < numPaginas; i++) {
   encontrada[i] = busca(“palavra”,paginas[i]);
}
```

Programação Paralela Estruturada

A “programação paralela estruturada” (sequencial) é composta por **padrões paralelos ou esqueletos algoritmicos**:

- Fork-Join, Map, Reduction, Scan, Stencil etc.

Estes padrões paralelos capturam estruturas comuns a programas paralelos.

- Elimina o uso explícito de threads.
- Remove (relaxa) as restrições de serialização dos padrões sequenciais.

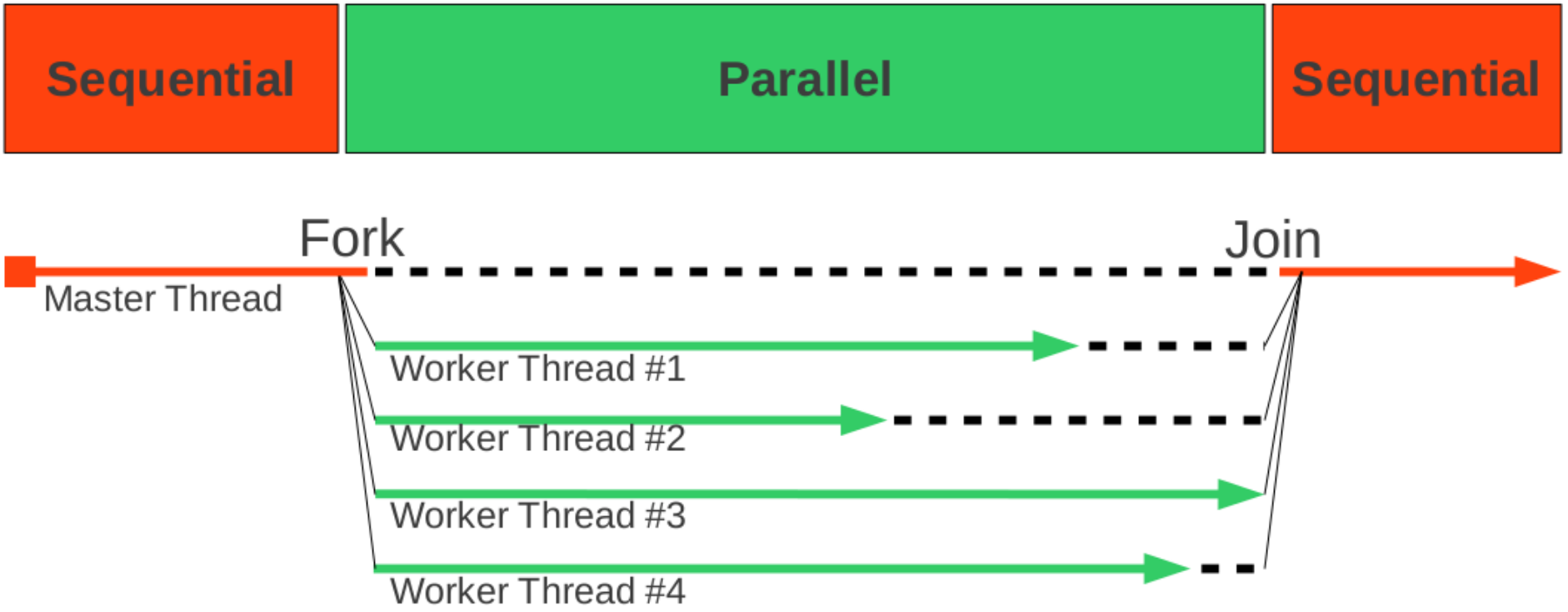
Fork-Join

É o padrão paralelo mais simples.

Permite que uma mesma seção de código seja executada em paralelo por várias threads trabalhadoras.

Uma thread mestre dispara **(fork)** várias threads trabalhadoras e ao final da execução de cada thread, elas sincronizam-se, restando apenas a thread mestre **(join)**.

Fork-Join


Sequential
Parallel
Sequential
Fork
Join
Master Thread
Worker Thread #1
Worker Thread #2
Worker Thread #3
Worker Thread #4

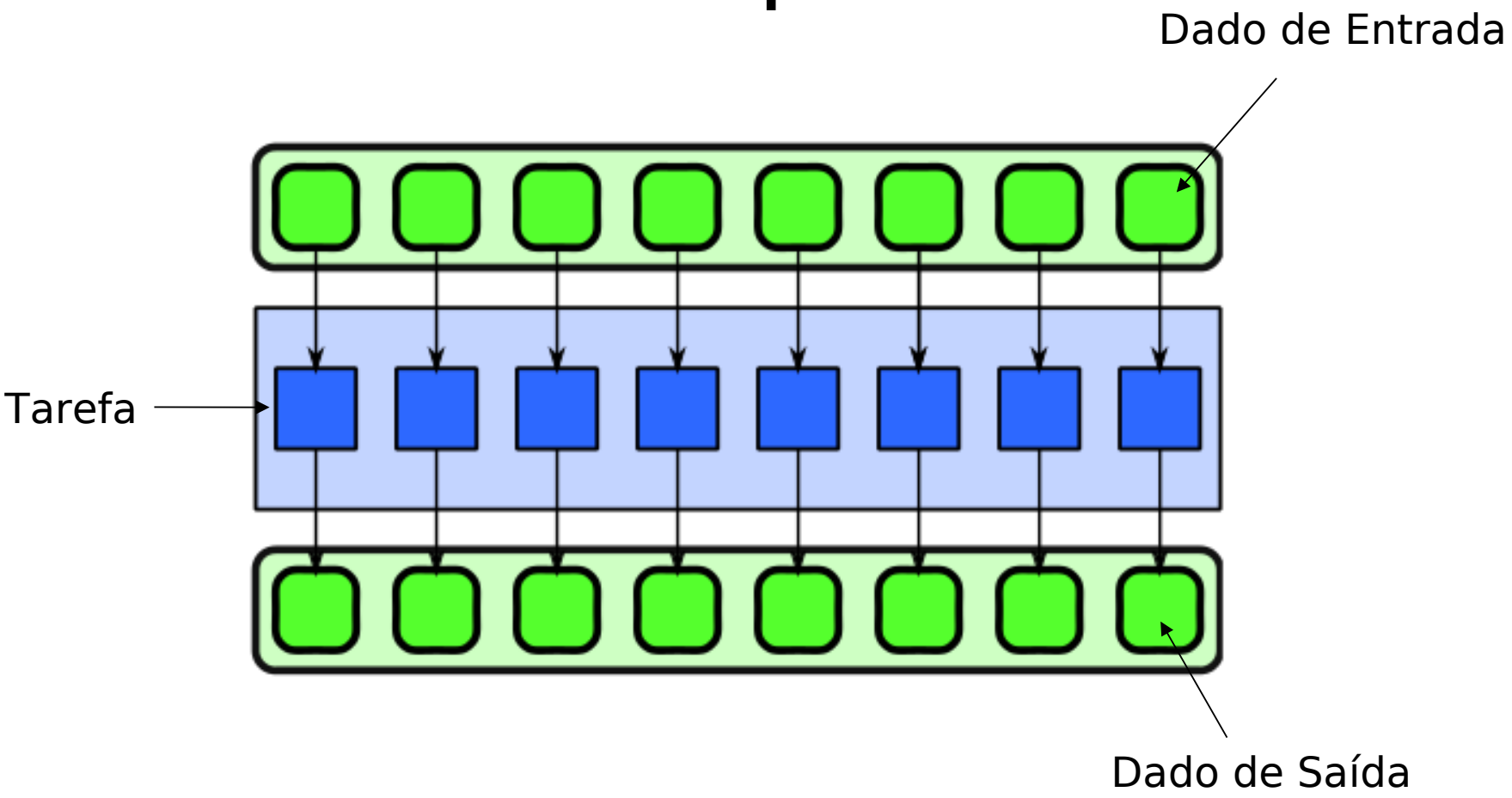
Map

O padrão **MAP** replica uma função sobre cada elemento de um conjunto de índices.

Pode ser considerado a estrutura de repetição sem a restrição de ordem de execução das iterações.

- É necessário que as funções (iterações) sejam independentes.
- Ex: parallel_for

Map

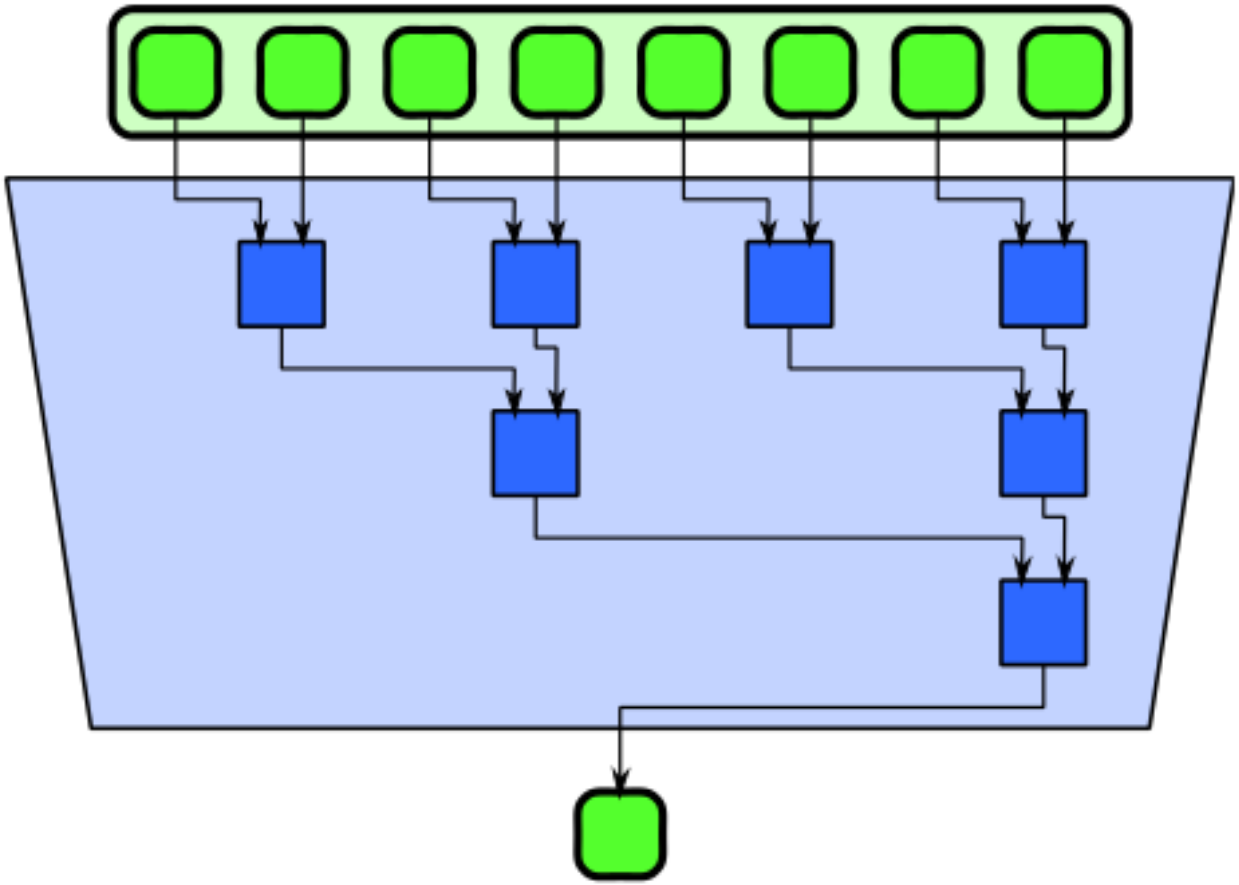


Reduction

O padrão **REDUCTION** combina cada elemento de uma coleção em um elemento usando um operador.

Exemplos: realizar um somatório de inteiros ou encontrar o maior valor de um vetor de inteiros.

Reduction



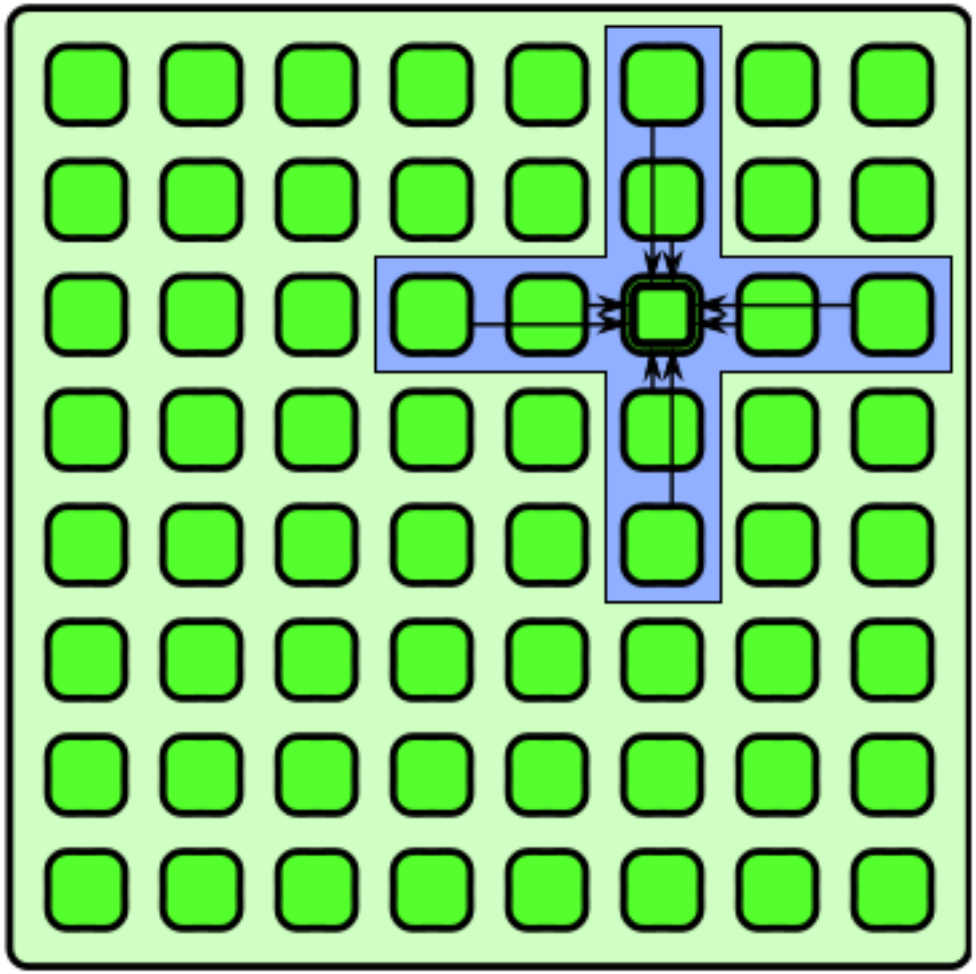
Stencil

O padrão **STENCIL** é uma generalização do MAP, onde cada função acessa não somente um único elemento, mas também seus vizinhos.

- Condições de borda precisam ser checadas.

Exemplo: convolução de imagens, detecção de quinas etc.

Stencil



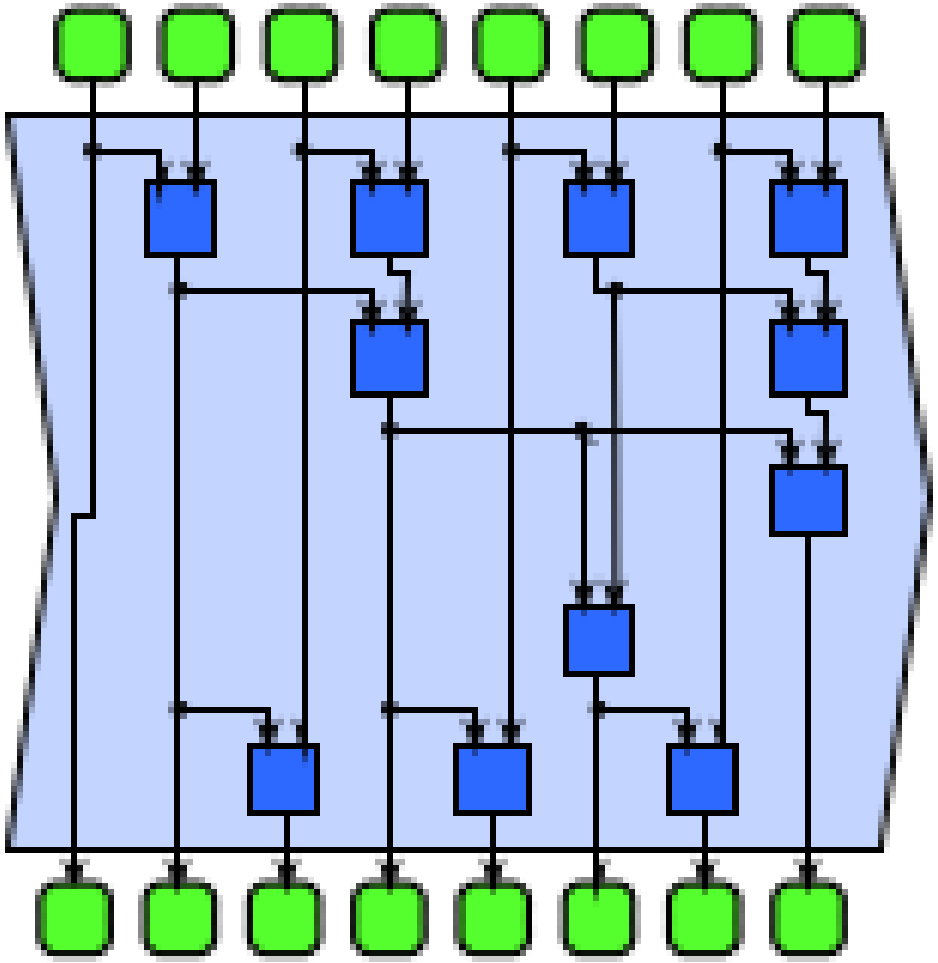
Scan

O padrão **SCAN** computa todas as reduções parciais de uma coleção de elementos, ou seja, para cada saída, uma redução é computada até este elemento.

Exemplo: somatórios parciais.

```
for (i = 1; i < n; i++) {
      a[i] += a[i-1];
}
```

Scan



Semântica e Implementação

Semântica

- O que o padrão faz?
- É apenas a funcionalidade do padrão (visão de fora).

Implementação

- Como o padrão executa na prática?
- Várias implementações são possíveis.
- Visão de dentro do padrão.

Suporte aos Padrões

OpenMP

- Map, Reduction e Fork-Join

CUDA / OpenCL

- Map

TBB

- Map, Reduction, Fork-Join e Scan

Nenhuma ferramenta suporta diretamente Stencil.