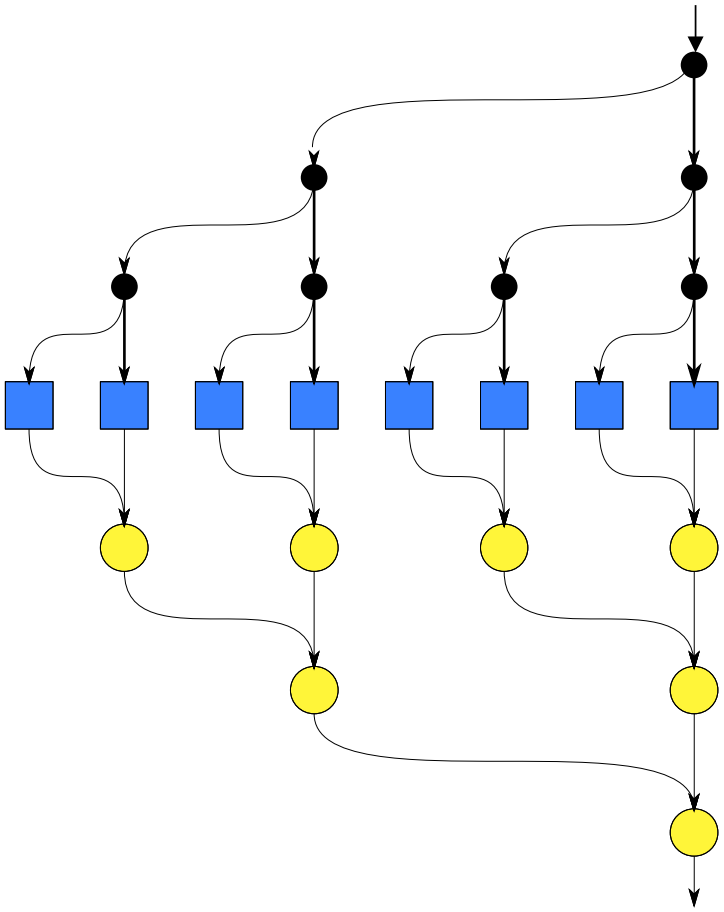
Padrões paralelos para controle de fluxo

- Estendem o paradigma sequencial com o objetivo de paralelizar o controle de fluxo.
- Os padrões apresentados nesta seção são:
  - *Fork-join*
  - *Map*
  - *Stencil*
  - *Reduction*
  - *Scan*

Fork-join

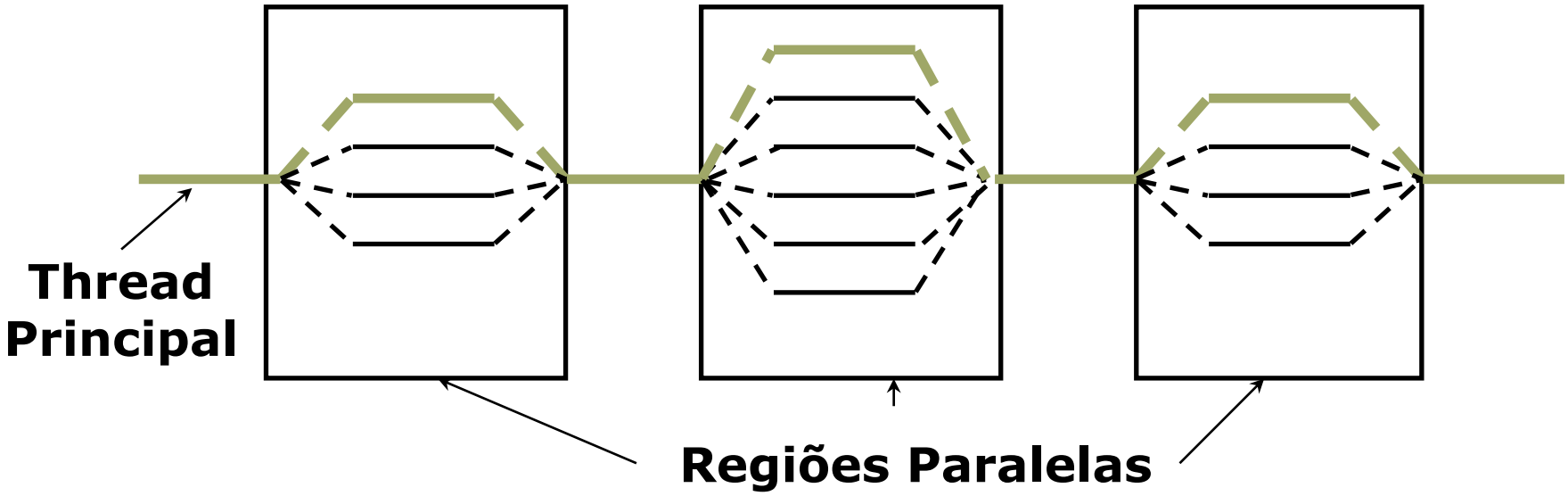
- Permite que o fluxo de controle seja dividido em múltiplos fluxos de execução paralelos.
  - Operação *fork*
- Estes fluxos se reunirão novamente no final de suas execuções, quando apenas um deles continuará.
  - Operação *join*
- Estas operações podem ser aninhadas.

Fork-join



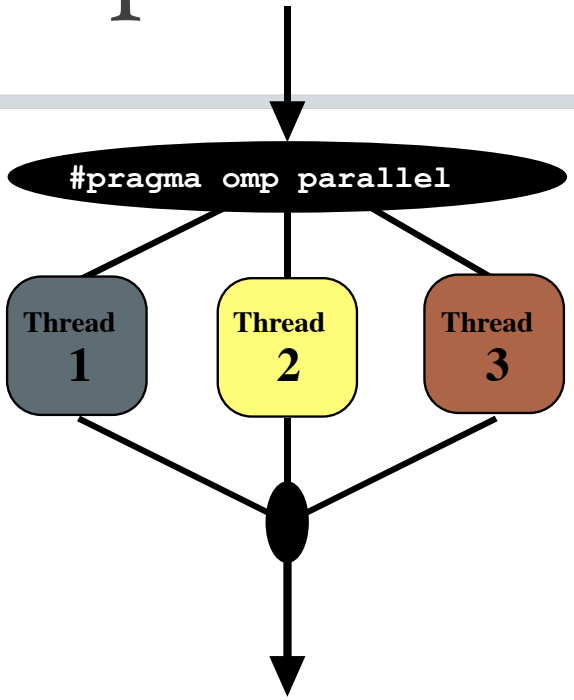
Exemplo 1: modelo de programação em OpenMP

- Paralelismo *Fork-join*:
  - A thread principal cria um time de threads conforme a necessidade.



Regiões Paralelas em OpenMP

- Define uma região paralela através de um bloco estruturado de código.
- As threads são criadas em paralelo.
- As threads ficam bloqueadas no final da execução.
- Os dados globais são compartilhados entre as threads.



C/C++ :

```
#pragma omp parallel
    {
            block
    }
```

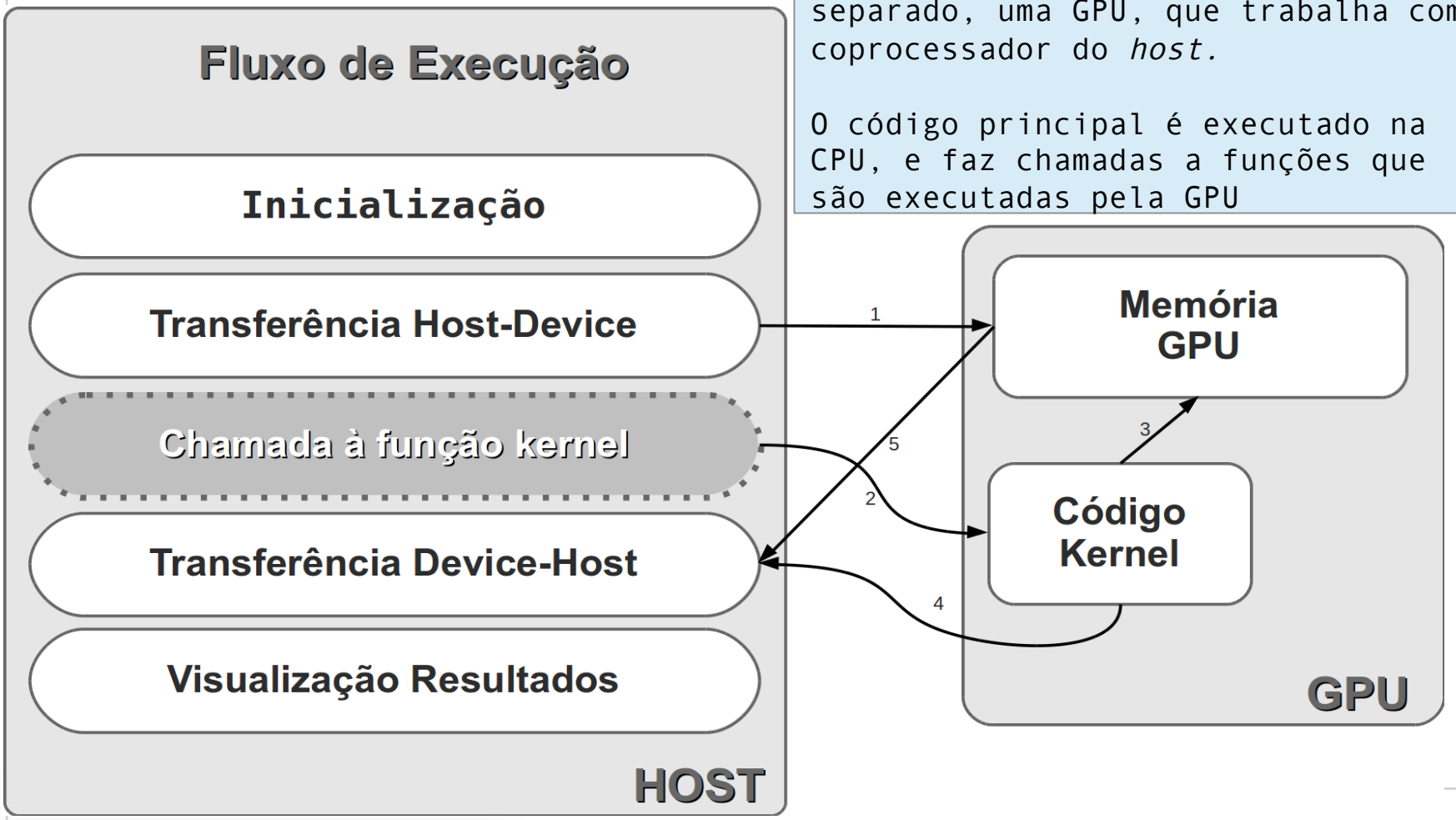
Exemplo: OpenMP

```
#include <stdio.h>
#include <omp.h>

int main (int argc, const char * argv[]) {

   omp_set_num_threads(4);

   #pragma omp parallel
      printf("Hello, World from thread %d!\n", omp_get_thread_num());

    return 0;
}
```

Exemplo 2: CUDA

O modelo assume que suas *threads* são executadas em um dispositivo separado, uma GPU, que trabalha como coprocessador do *host*.

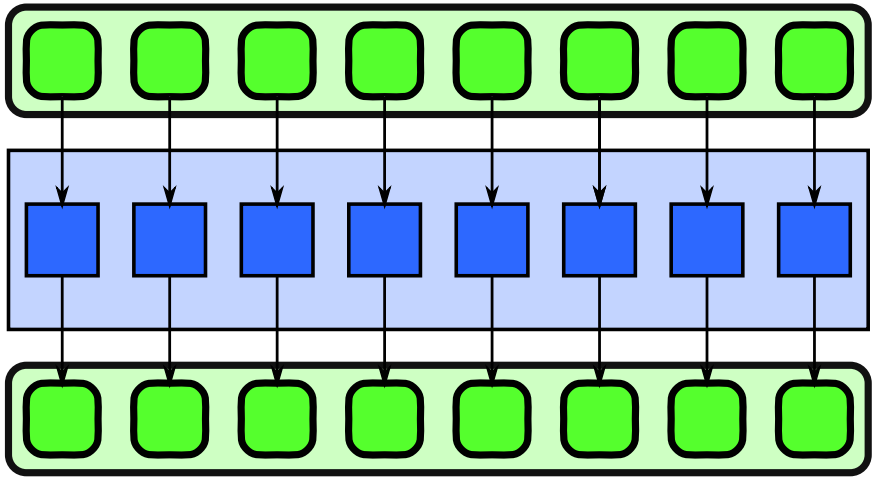
O código principal é executado na CPU, e faz chamadas a funções que são executadas pela GPU



Map

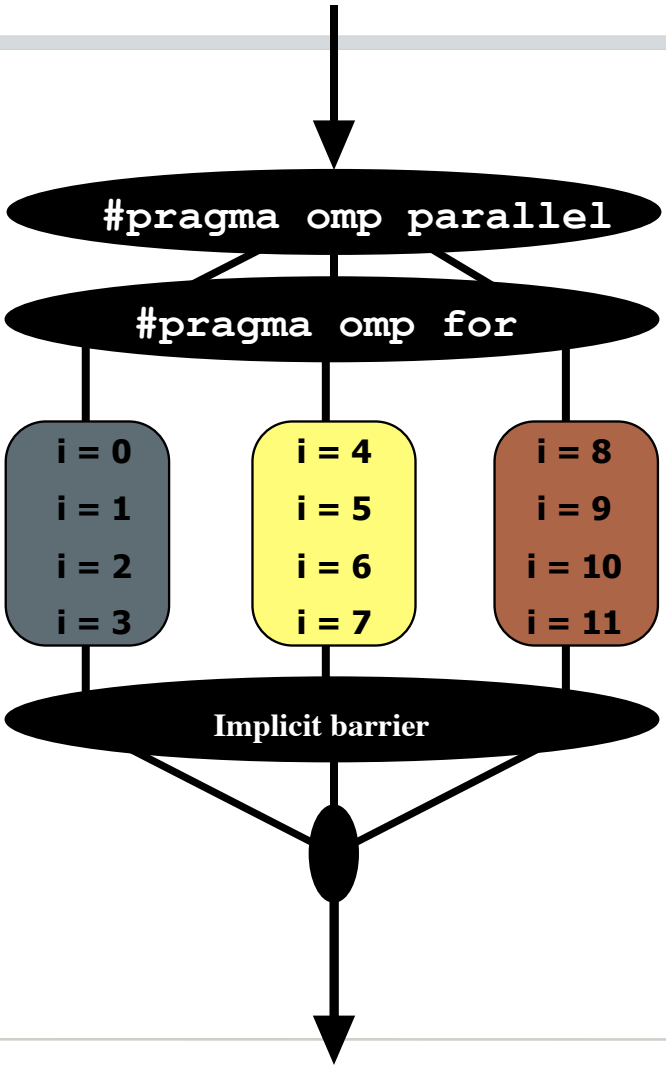
- Replica uma mesma operação sobre um conjunto de elementos indexados.
- O conjunto de índices pode ser abstrato ou estar diretamente relacionado aos elementos de uma coleção.
- Este padrão se aplica à paralelização de laços, nos casos onde se pode aplicar uma função independente a todo o conjunto de elementos.

Map



Exemplo 1: laços em OpenMP

```
#pragma omp parallel
#pragma omp for
   for(i = 0; i < 12; i++)
      c[i] = a[i] + b[i]
```



Exemplo 2: CUDA

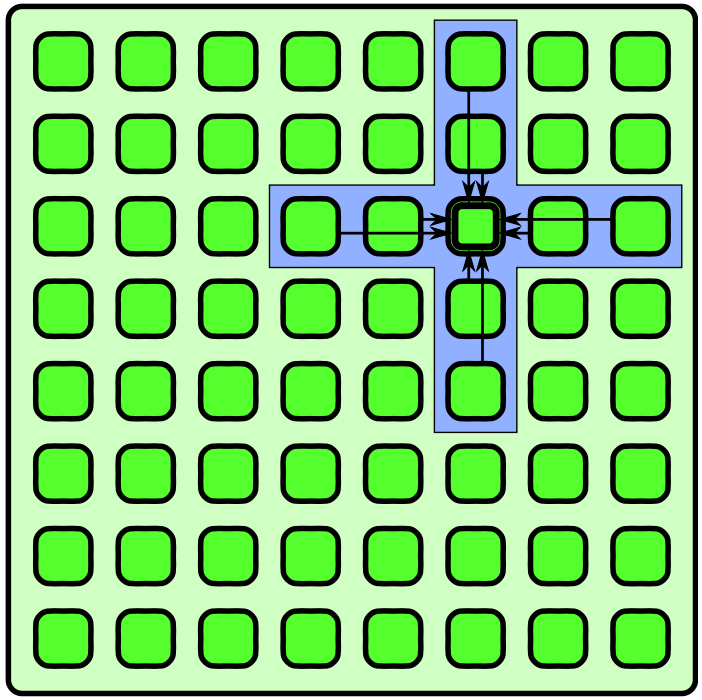
- *Threads* executam o código definido em uma função *kernel*.
- Uma chamada a uma função *kernel*, dispara a execução de N instâncias paralelas por N *threads* CUDA.

```
01 // Definição da Função Kernel.
02 __global__ void VecAdd(float* A, float* B, float* C) {
03     int i = threadIdx.x;
04     C[i] = A[i] + B[i];
05 }
06 int main() { ... // Invocação do Kernel com N threads.
07   VecAdd<<<1, N>>>(A, B, C);
08 …
09 }
```

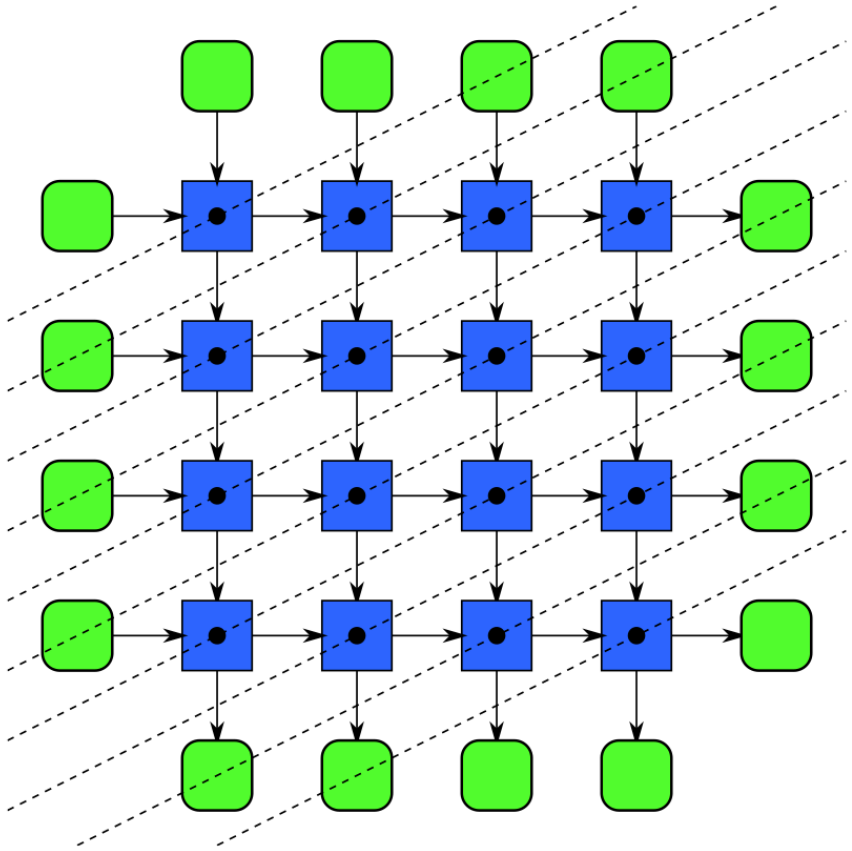
Stencil

- É uma generalização do padrão *map*, onde a função é aplicada sobre um conjunto de vizinhos.
- Os vizinhos são definidos a partir de um conjunto *offset* relativos a cada ponto do conjunto.
- Todos podem ser calculados em paralelo, porém alguns cuidados de implementação devem ser tomados para evitar o não-determinismo (assim como no padrão *map*).

Stencil



Exemplo – Dependência de Dados em Computação Sistólica



Exemplo – Dependência de Dados em Computação Sistólica

```
1  void my_recurrence(
2      size_t v,          // number of elements vertically
3      size_t h,          // number of elements horizontally
4      const float a[v][h], // input 2D array
5      float b[v][h]      // output 2D array (boundaries already initialized )
6  ) {
7      for (int i=1; i<v; ++i)
8          for (int j=1; j<h; ++j)
9              b[i][j] = f(b[i-1][j], b[i][j-1], a[i][j]);
10 }
```

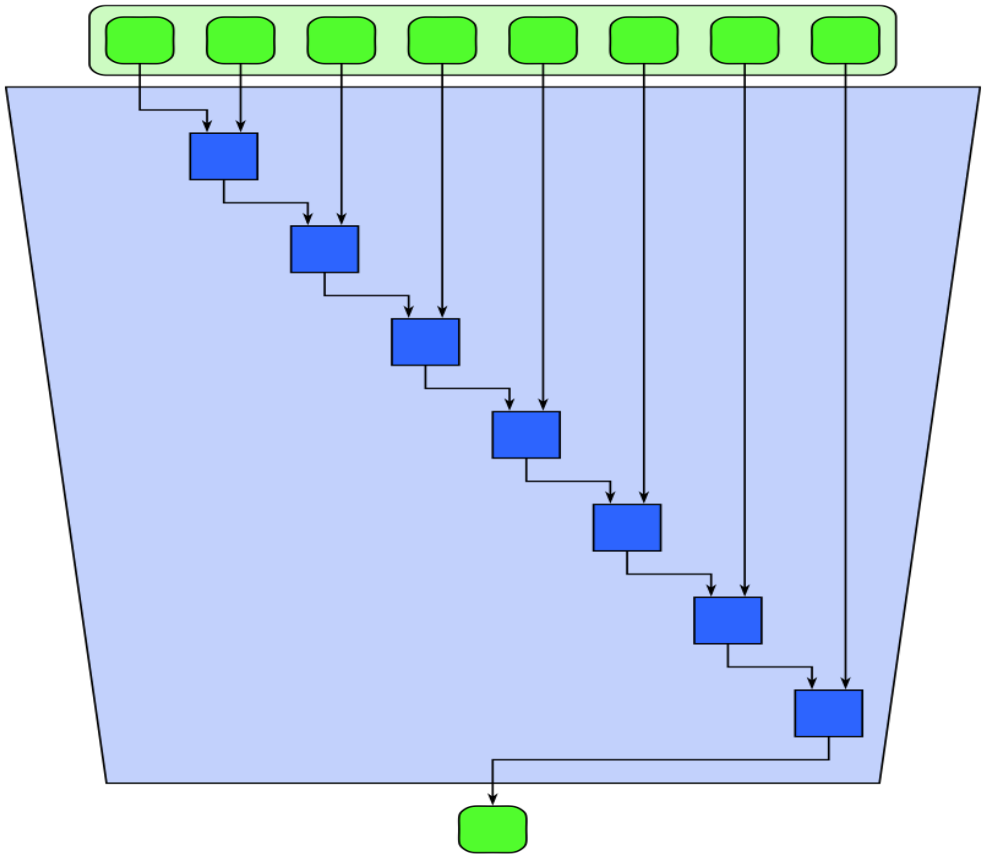
Reduction

- Combina todos os elementos de uma coleção em um único elemento a partir de uma função combinadora associativa.
- Dada a associatividade da operação, muitas ordens de avaliação podem ser implementadas.
- Uma operação muito comum em aplicações numéricas é realizar a soma ou encontrar o máximo de um conjunto de elementos.

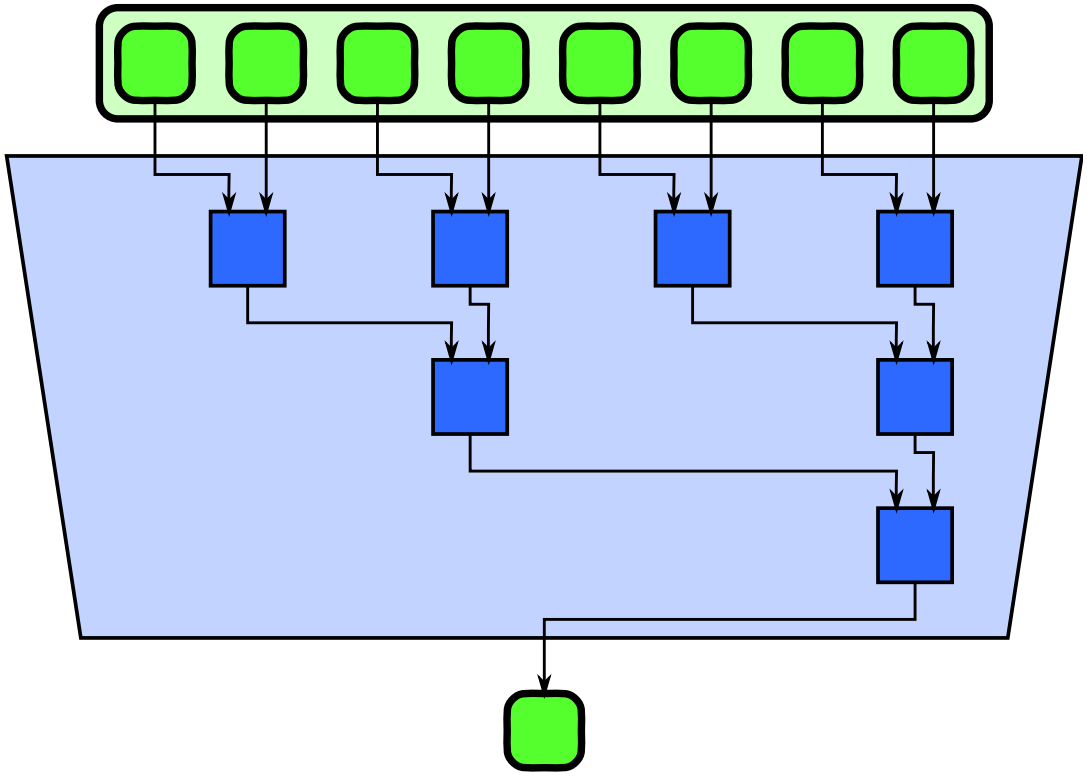
Exemplo – Soma Sequencial

```
1  double my_add_reduce(
2    const double a[], // input array
3    size_t n          // number of elements
4  ) {
5    double r = 0.0; // initialize with the identity for addition
6    for (int i = 0; i < n; ++i)
7      r += a[i];  // each iteration depends on the previous one
8    return r;
9  }
```

Esquema – Soma Sequencial



Padrão – Reduction em Árvore

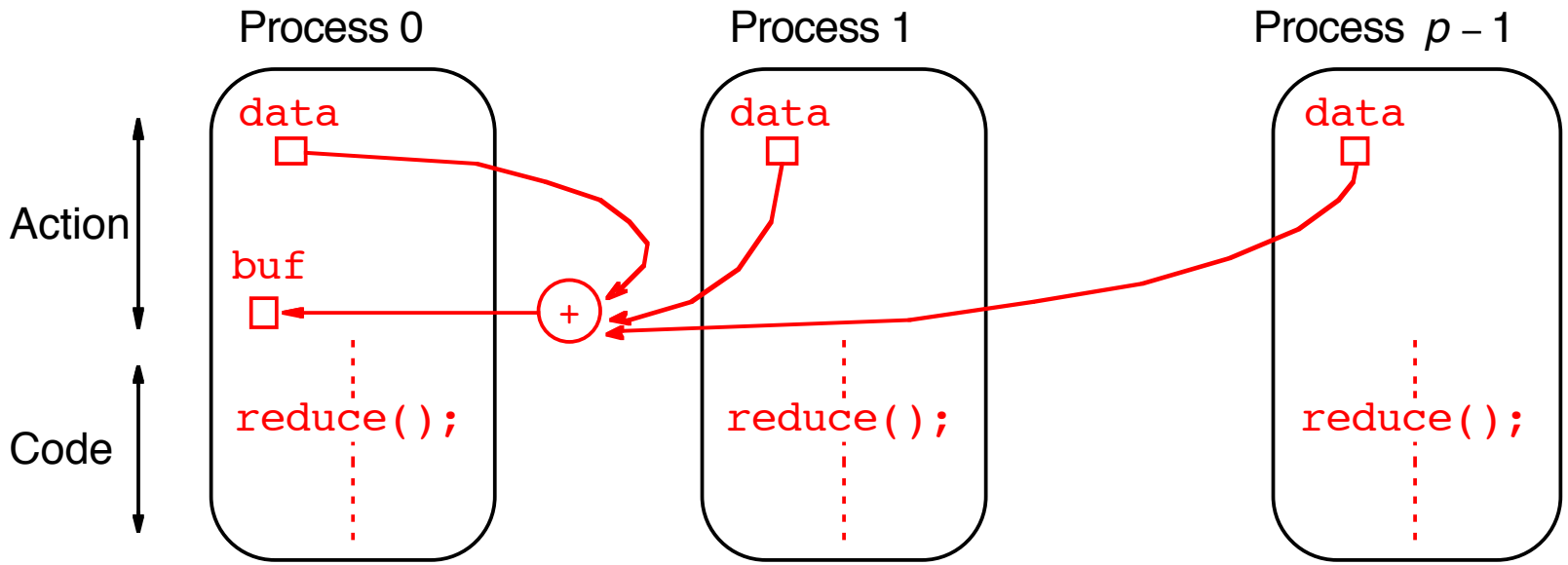


Exemplo: MPI

- MPI (*Message Passing Interface*)
  - Padrão para bibliotecas de programação por passagem de mensagens (várias implementações conhecidas).
  - Usada em programação onde não há memória compartilhada (exemplo: clusters).
- Possui um conjunto de rotinas de comunicação coletiva.
  - Implementa os padrões ***reduction***, ***scan***, ***gather*** e ***scatter***.

Exemplo em MPI: *Reduce*

- Realiza uma coleta no *root*, combinando os valores através de uma operação lógica ou aritmética.
- O método de redução paralela empregado é implementado pela biblioteca e é transparente ao programador.



Exemplo: *reduction* em MPI

- Somatório de um *array* de elementos:

```
MPI_Bcast(data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD);

x = n/nproc; /* Add my portion Of data */
low = myid * x;
high = low + x;
for(i = low; i < high; i++)
      myresult += data[i];

MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);

if (myid == 0) printf("The sum is %d.\n", result);
```

Exemplo em OpenMP: cláusula *reduction*

reduction (op : list)

- As variáveis em "list" devem ser *shared* na região paralela

- Dentro da construção paralela
  - Uma cópia PRIVATE de cada variável é criada e inicializada dependendo da operação
  - Essas cópias são atualizadas localmente pelas threads
  - Ao final da construção, as cópias locais são combinadas através da operação "op" em um único valor, e combinado com o valor original da variável SHARED.

Exemplo em OpenMP: cláusula *reduction*

```
#pragma omp parallel for reduction(+:sum)
   for(i=0; i<N; i++) {
     sum += a[i] * b[i];
   }
```

- Uma cópia local de ***sum*** para cada *thread*
- Todas as cópias locais de sum são adicionadas e armazenadas na sua variável "global"

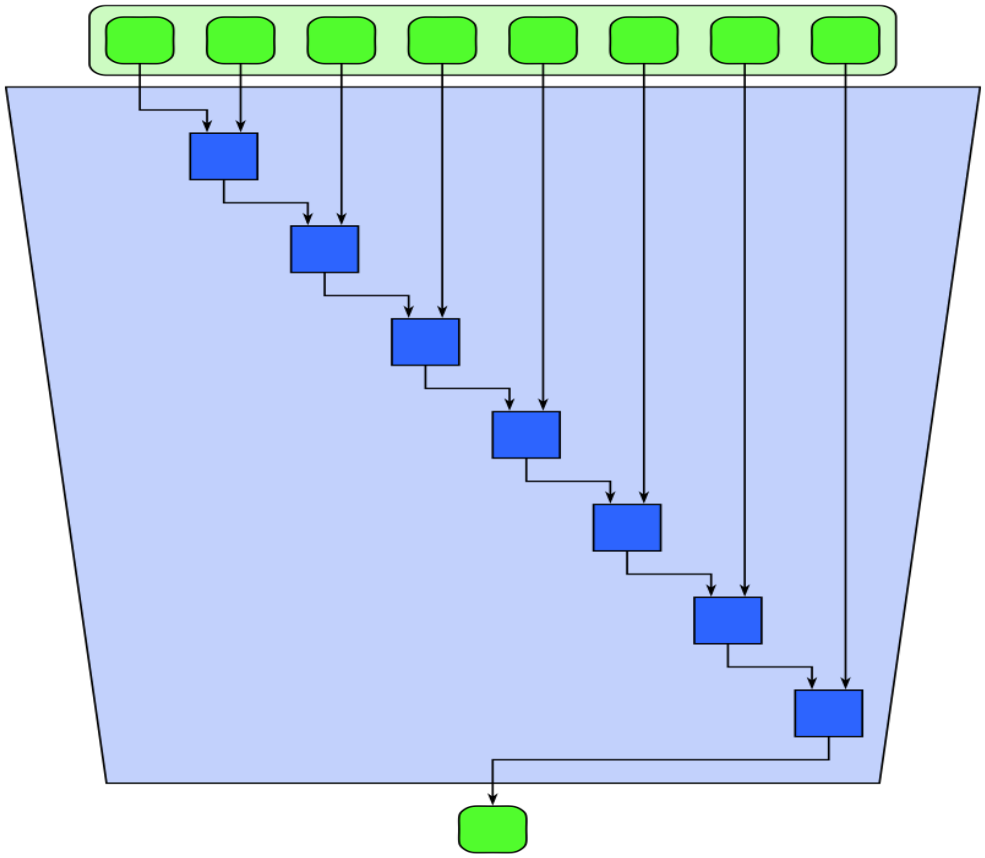
Scan

- Calcula todas as reduções parciais de um conjunto.
- Para cada saída do conjunto, uma redução parcial até sua posição neste conjunto é calculada.
  - Ou seja, uma redução com todos os valores anteriores de cada elemento do conjunto indexado.
- Aplicações incluem integração de funções, geração de números aleatórios, análise de séries temporais.

Exemplo – Soma Sequencial

```
1  void my_add_iscan(
2    const float a[], // input array
3    float b[],     // output array
4    size_t n       // number of elements
5  ) {
6    if (n>0) b[0] = a[0]; // equivalent to assuming b[i−1] is zero
7    for (int i = 1; i < n; ++i)
8      b[i] = b[i−1] + a[i]; // each iteration depends on the previous one
9  }
```

Esquema – Soma Sequencial



Padrão - Scan