Padrão Scan em CUDA

Padrão Scan

Dado um vetor de entrada, o padrão scan gera um vetor de saída onde o valor de cada elemento é o somatório de todos os seus elementos predecessores.

Sequencialmente ele é geralmente implementado em O(N)

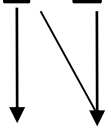
- Exemplo: somatórios parciais

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

Exemplo

Realizar o scan no vetor A.

A [1 2 5 7 9 6]



A [1 3 8 15 24 30]

Implementação Paralela

Pode-se realizar várias operações em paralelo com uma determinado vizinho a uma certa distância (*stride*). Ao final de **log n** passos, as somas parciais são computadas.

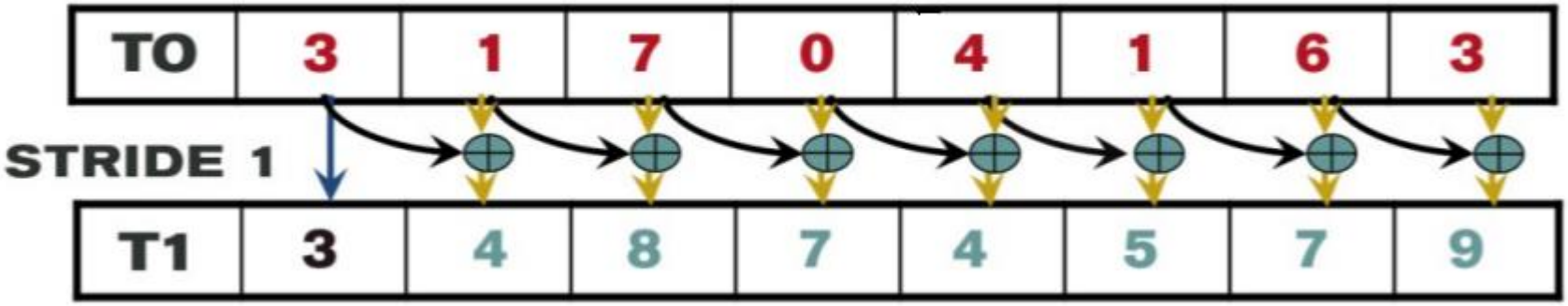
A cada passo, metade das threads não são mais necessárias.

Cada bloco armazena um resultado parcial, que devem ser processados pelo **host** e adicionados a todos os elementos.

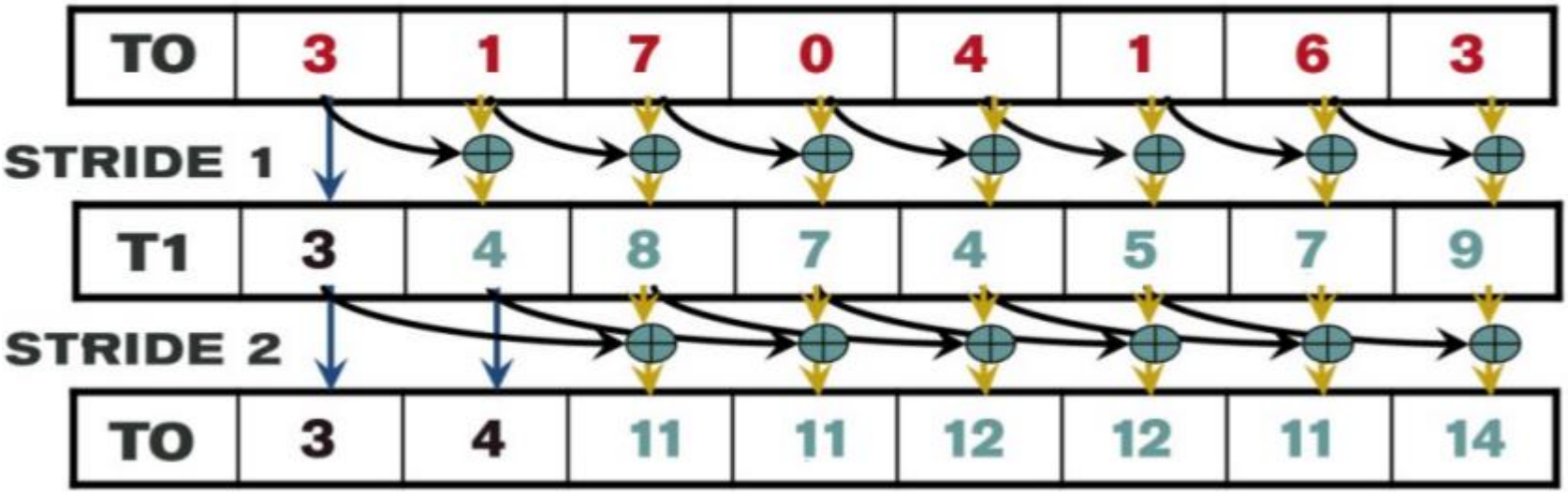
Implementação Paralela

TO	3	1	7	0	4	1	6	3

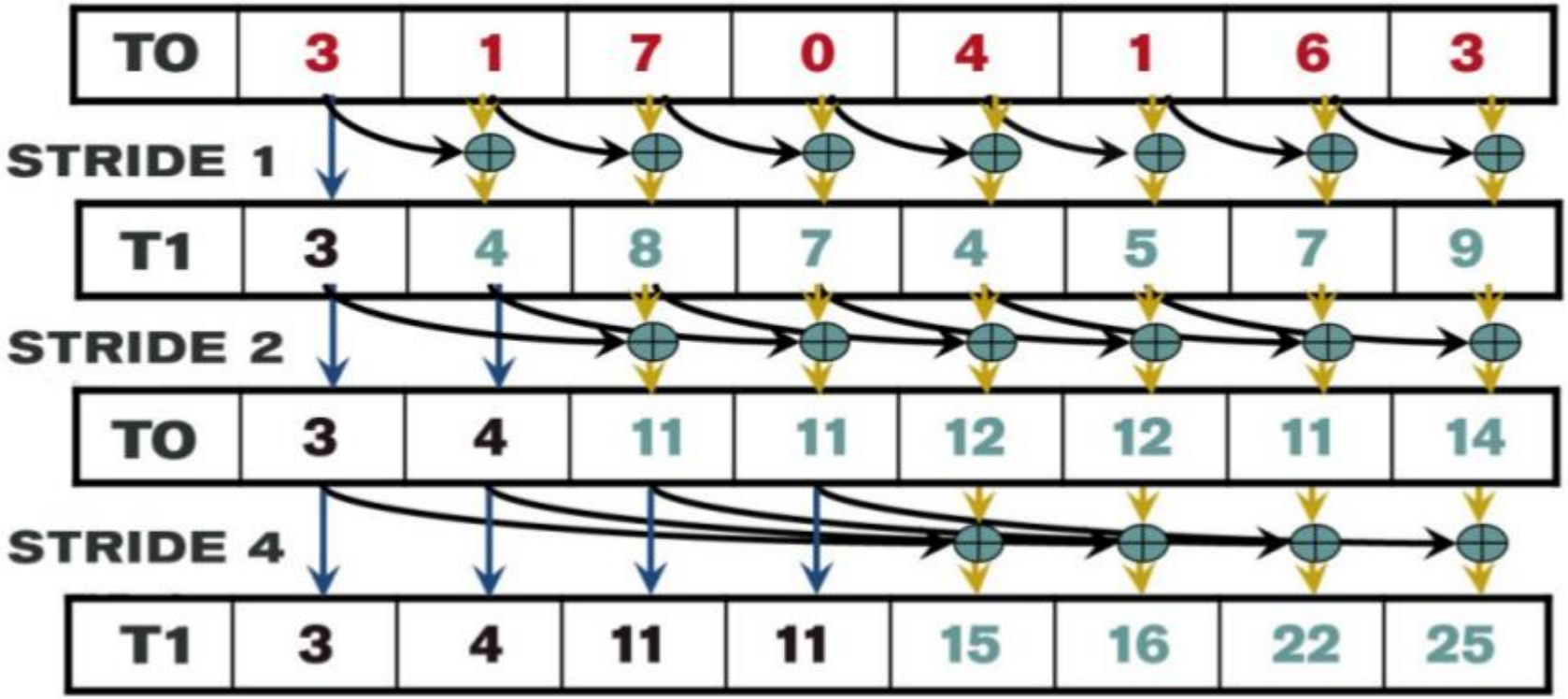
Implementação Paralela



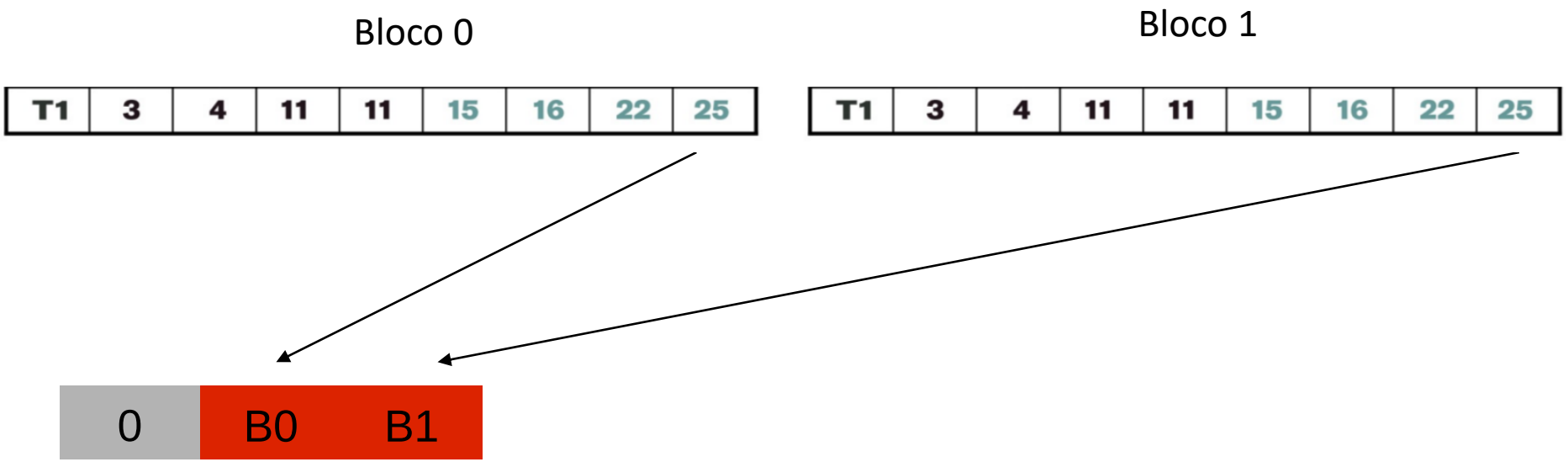
Implementação Paralela



Implementação Paralela



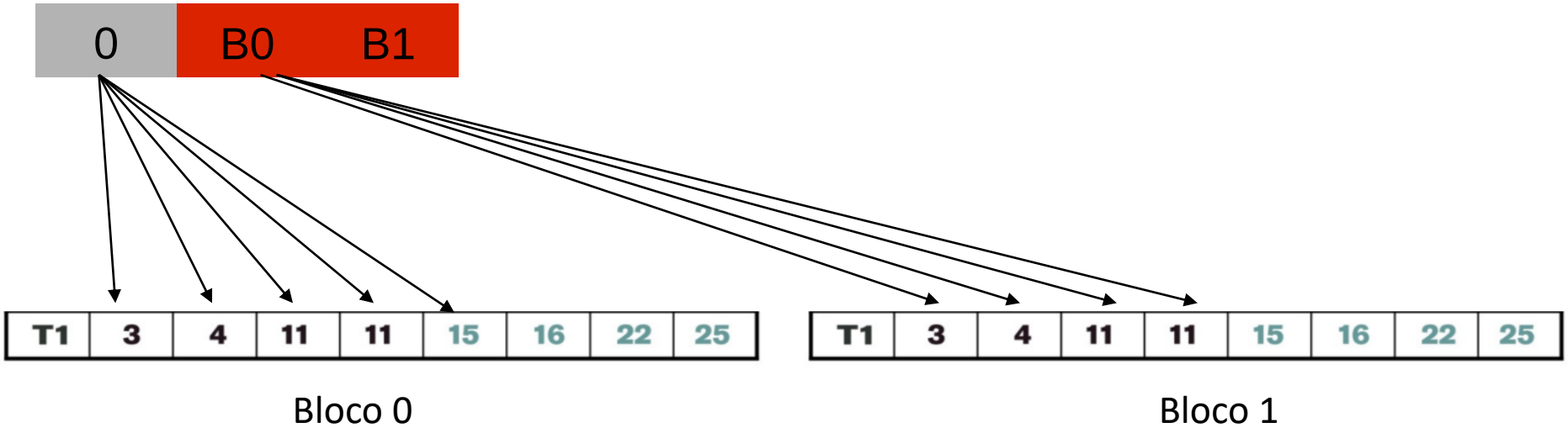
Paralelização dos Blocos




```
s[0] = 0;

for (int i = 1; i < num_blocks; i++)
   s[i] += s[i-1];
```

Propagação dos Somatórios



Algoritmo do Scan em CUDA

Copia vetor de entrada A para o device.

Executa o kernel do scan no device.

Copia o vetor de saída S para o host.

Realiza o scan no vetor de saída S no host.

Copia o vetor de saída S para o device.

Executa o kernel de soma no device.

Copia o vetor de entrada A para o host.

Kernel Scan - Carregar Elementos

```
__global__ void scan_cuda(double* a, double *s, int width) {
  int t = threadIdx.x;
  int b = blockIdx.x*blockDim.x;
  double x;

  // cria vetor na memória local
  __shared__ double p[1024];

  // carrega elementos do vetor da memória global para a local
  if(b+t < width)
    p[t] = a[b+t];

  // espera que todas as threads tenham carregado seus elementos
  __syncthreads();
```

Kernel Scan – Somatório Parcial

```
for (int i = 1; i < blockDim.x; i *= 2) {  // realiza o scan em log n passos
  if(t >= i)  // verifica se a thread ainda participa neste passo
    x = p[t] + p[t-i];  // atribui a soma para uma variável temporária

  __syncthreads(); // espera threads fazerem as somas

  if(t >= i)
    p[t] = x;  // copia a soma em definitivo para o vetor local

  __syncthreads();
 }

 if(b + t < width) // copia da memória local para a global
  a[b+t] = p[t];

 if(t == blockDim.x-1)  // se for a última thread do bloco
  s[blockIdx.x+1] = a[b+t];    // copia o seu valor para o vetor de saída
}
```

Kernel Adição

Após calculadas as somas parciais dentro de cada bloco, estes valores devem ser somados (scan feito no host) e propagados para todos os elementos.

```
__global__ void add_cuda(double *a, double *s, int width) {
  int t = threadIdx.x;
  int b = blockIdx.x*blockDim.x;

  // soma o somatório do último elemento do bloco anterior ao elemento atual
  if(b+t < width)
    a[b+t] += s[blockIdx.x];
}
```