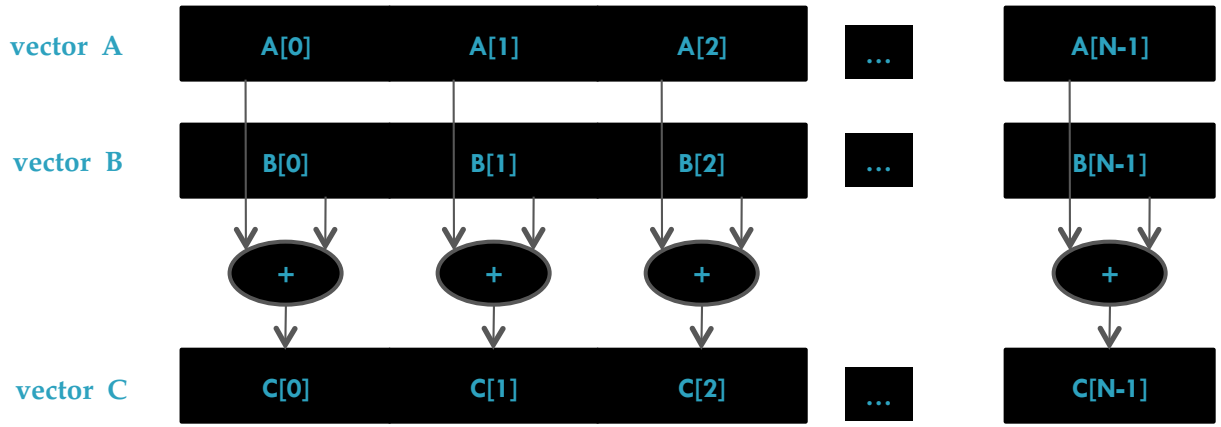
Objective

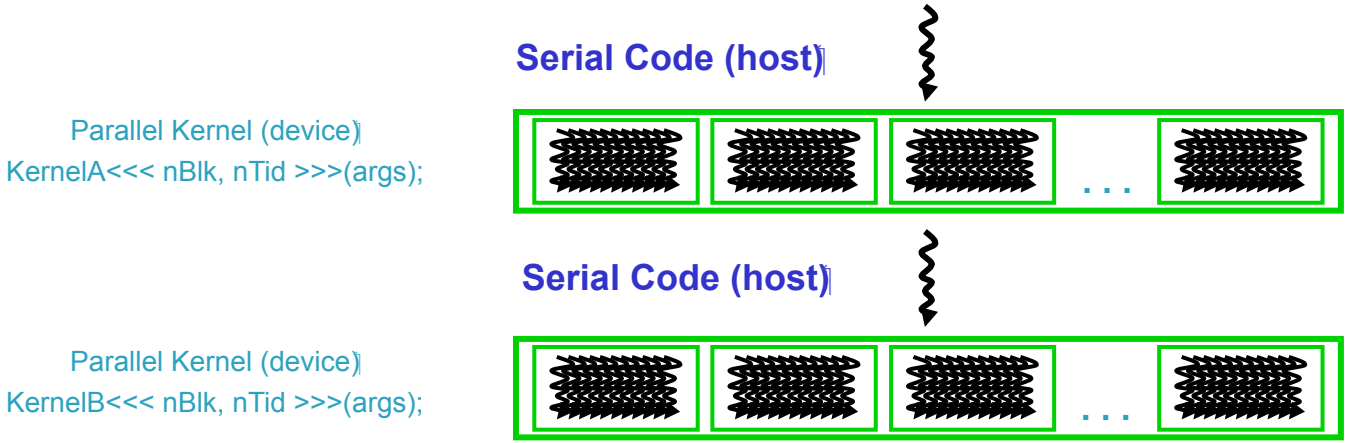
- To learn about CUDA threads, the main mechanism for exploiting of data parallelism
  - Hierarchical thread organization
  - Launching parallel execution
  - Thread index to data index mapping

Data Parallelism - Vector Addition Example



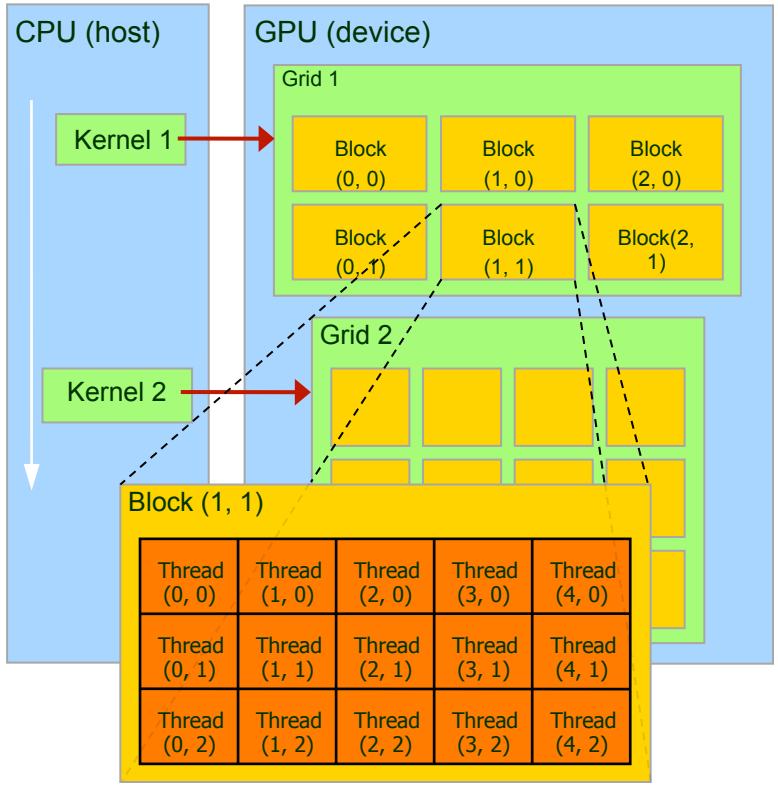
CUDA Execution Model

- Heterogeneous host (CPU) + device (GPU) application C program
  - Serial parts in **host** C code
  - Parallel parts in **device** SPMD kernel code



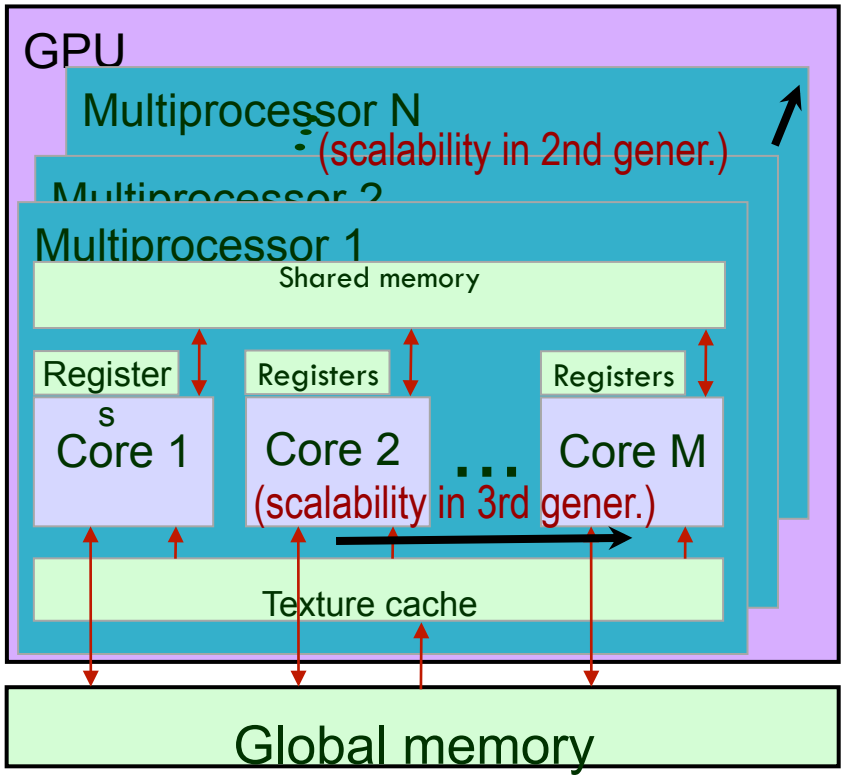
Partitioning data and computations

- A block is a batch of threads which can cooperate by:
- Sharing data via shared memory.
- Synchronizing their execution for hazard-free memory accesses.
- A kernel is executed as a 1D or 2D grid of 1D, 2D or 3D of thread blocks.
- Multidimensional IDs are very convenient when addressing multidimensional arrays, for each thread has to bound its area/volume of local computation.



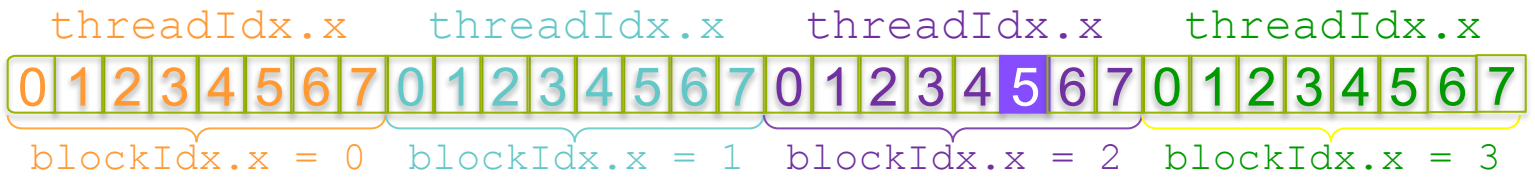
Running in parallel (regardless of hardware generation)

- `vecAdd <<< 1, 1 >>>`
  ( ) Executes 1 block composed of 1 thread - no parallelism.
- `vecAdd <<< B, 1 >>>`
  ( ) Executes B blocks composed on 1 thread. Inter-multiprocessor parallelism.
- `vecAdd <<< B, M >>>`
  ( ) Executes B blocks composed of M threads each. Inter- and intra-multiprocessor parallelism.



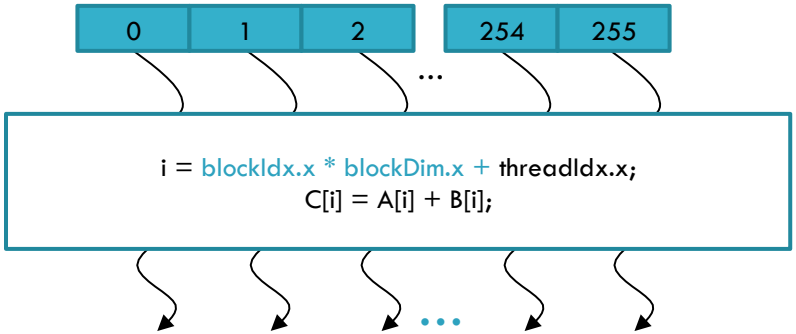
Indexing arrays with blocks and threads

Qual seria a configuração nesse caso?

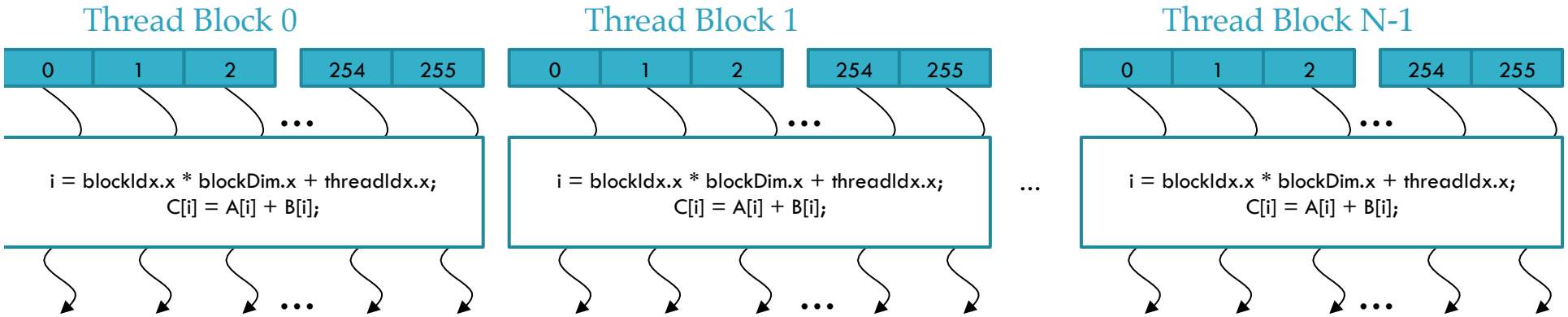


Arrays of Parallel Threads

- A CUDA kernel is executed by a grid (array) of threads
  - All threads in a grid run the same kernel code (Single Program Multiple Data)
  - Each thread has indexes that it uses to compute memory addresses and make control decisions



Thread Blocks: Scalable Cooperation

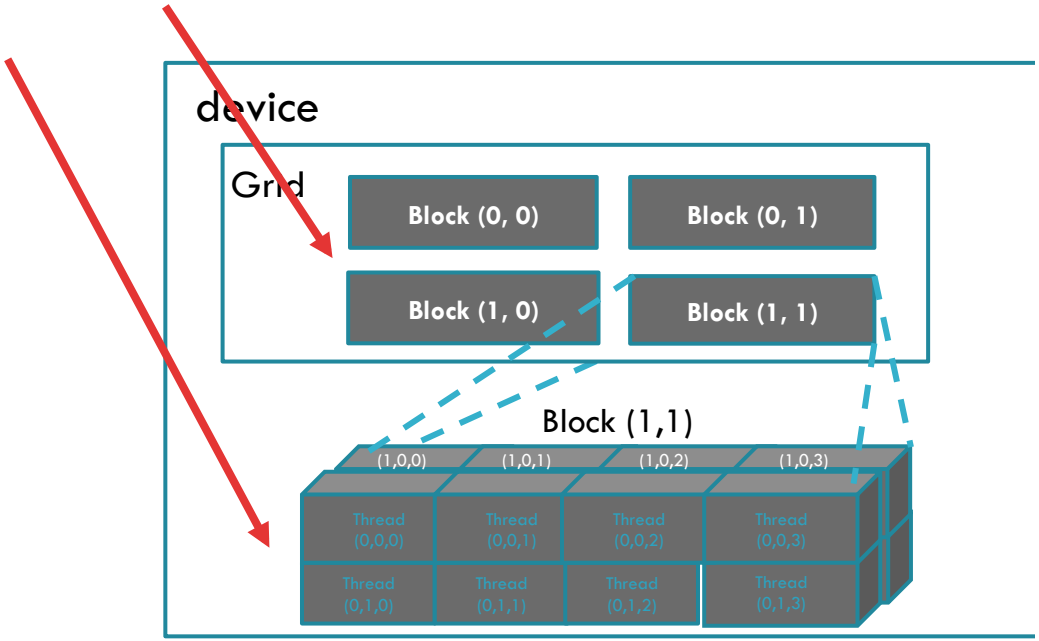


- Divide thread array into multiple blocks
  - Threads within a block cooperate via **shared memory, atomic operations** and **barrier synchronization**
  - Threads in different blocks do not interact

blockIdx and threadIdx

- Each thread uses indices to decide what data to work on
  - blockIdx: 1D, 2D, or 3D (CUDA 4.0)
  - threadIdx: 1D, 2D, or 3D

- Simplifies memory addressing when processing multidimensional data
  - Image processing
  - Solving PDEs on volumes
  - …



Handling arbitrary vector sizes

- Typical problems are not friendly multiples of blockDim.x, so we have to prevent accessing beyond the end of arrays:

```
// Add two vectors of size N: C[1..N] = A[1..N] + B[1..N]
__global__ void vecAdd(float* A, float* B, float* C, N) {
    int tid = threadIdx.x + (blockDim.x * blockIdx.x);
    if (tid < N)
      C[tid] = A[tid] + B[tid];
}
```

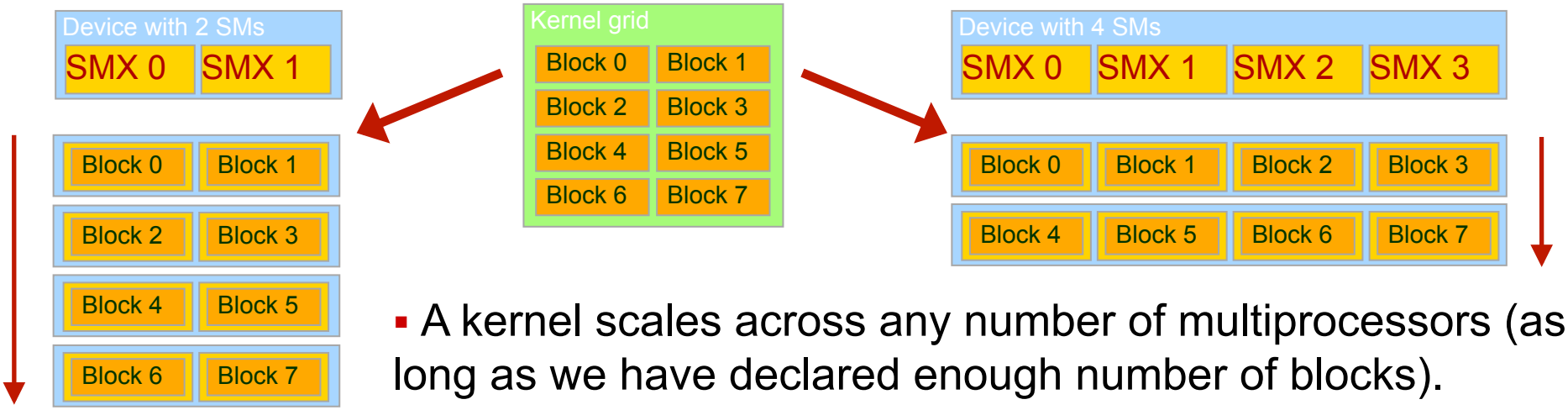
- And now, update the kernel launch to include the "incomplete" block of threads:

```
      vecAdd<<< (N + M-1)/256, 256>>>(d_A, d_B, d_C, N);
```

- Modifique seu trabalho para executar com um N arbitrário.

Transparent scalability

- Since blocks cannot synchronize:
- The hardware is free to schedule the execution of a block on any multiprocessor.
- Blocks may run sequentially or concurrently depending on resources



- A kernel scales across any number of multiprocessors (as long as we have declared enough number of blocks).