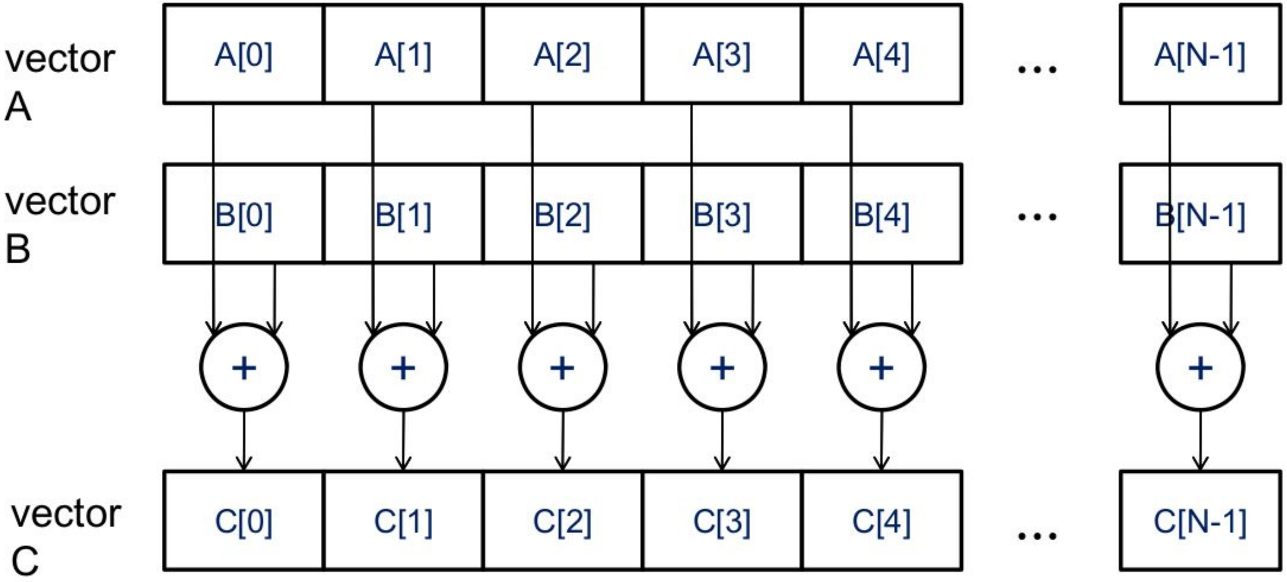
Introduction to Data Parallelism and CUDA C

John H. Osorio Ríos

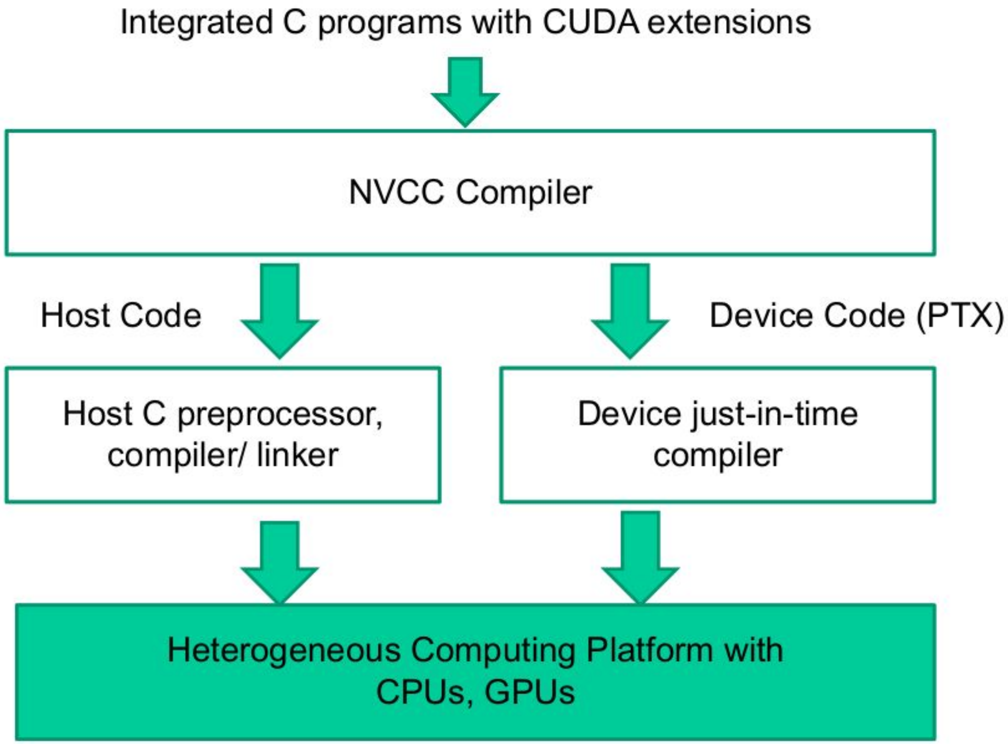
NVIDIA
CUDA™
TEACHING
CENTER

SCIENTIA ET TECHNICA PRO HVMANITATE SVB LIBERTATIS TVTELA
UT

Data Parallelism

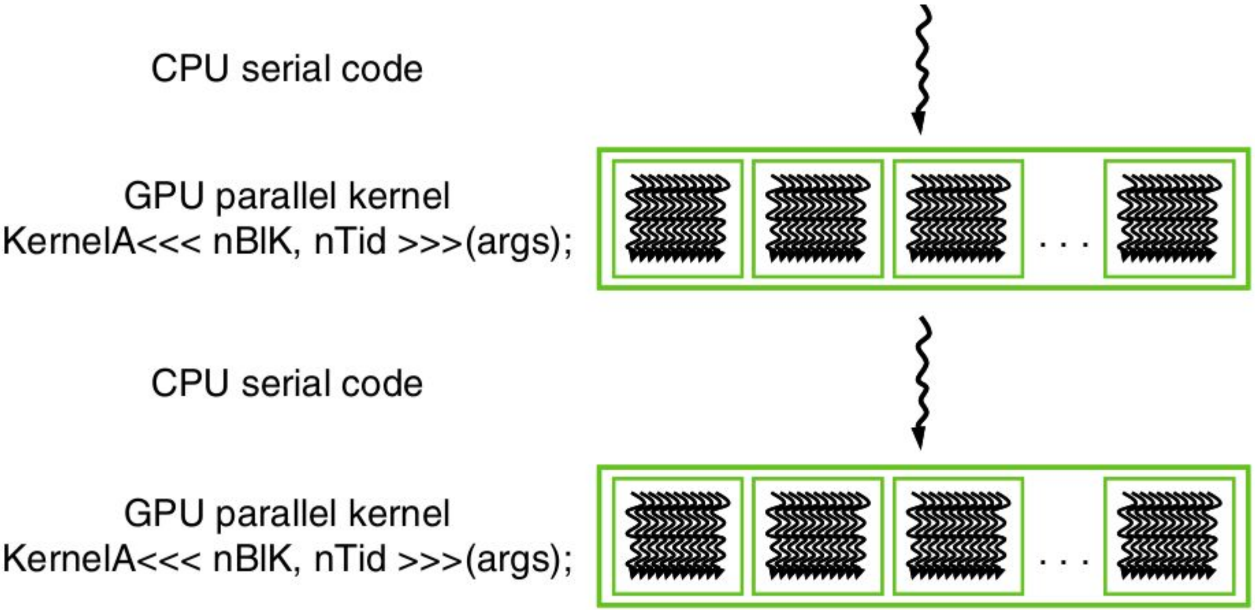


CUDA Program Structure


Integrated C programs with CUDA extensions
NVCC Compiler
Host Code
Device Code (PTX)
Host C preprocessor, compiler/ linker
Device just-in-time compiler
Heterogeneous Computing Platform with CPUs, GPUs

Compilation process

CUDA Program Structure



CPU serial code

GPU parallel kernel
KernelA<<< nBlK, nTid >>>(args);

CPU serial code

GPU parallel kernel
KernelA<<< nBlK, nTid >>>(args);

Execution of a CUDA program

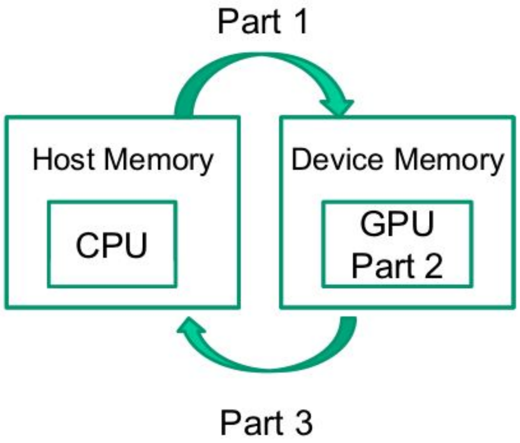
A Vector Addition Kernel

```
// Compute vector sum h_C = h_A+h_B
void vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
  for (i = 0; i < n; i++) h_C[i] = h_A[i] + h_B[i];
}

int main()
{
     // Memory allocation for h_A, h_B, and h_C
     // I/O to read h_A and h_B, N elements each
     …
     vecAdd(h_A, h_B, h_C, N);
}
```

Traditional vector addition

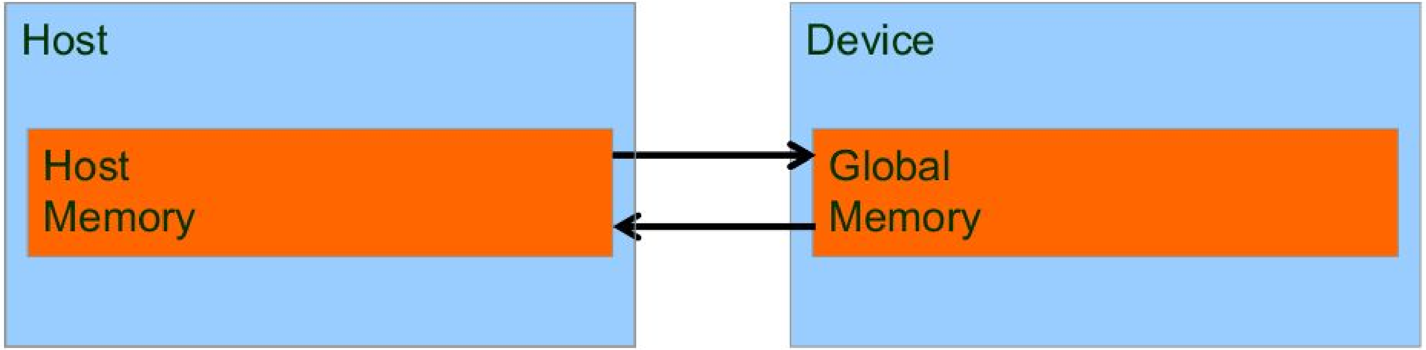
A Vector Addition Kernel

```
#include <cuda.h>
…
void vecAdd(float* A, float*B, float* C, int n)
{
   int size = n* sizeof(float);
    float *A_d, *B_d, *C_d;
   …
1. // Allocate device memory for A, B, and C
   // copy A and B to device memory

2. // Kernel launch code – to have the device
   // to perform the actual vector addition

3. // copy C from the device memory
   // Free device vectors
}
```



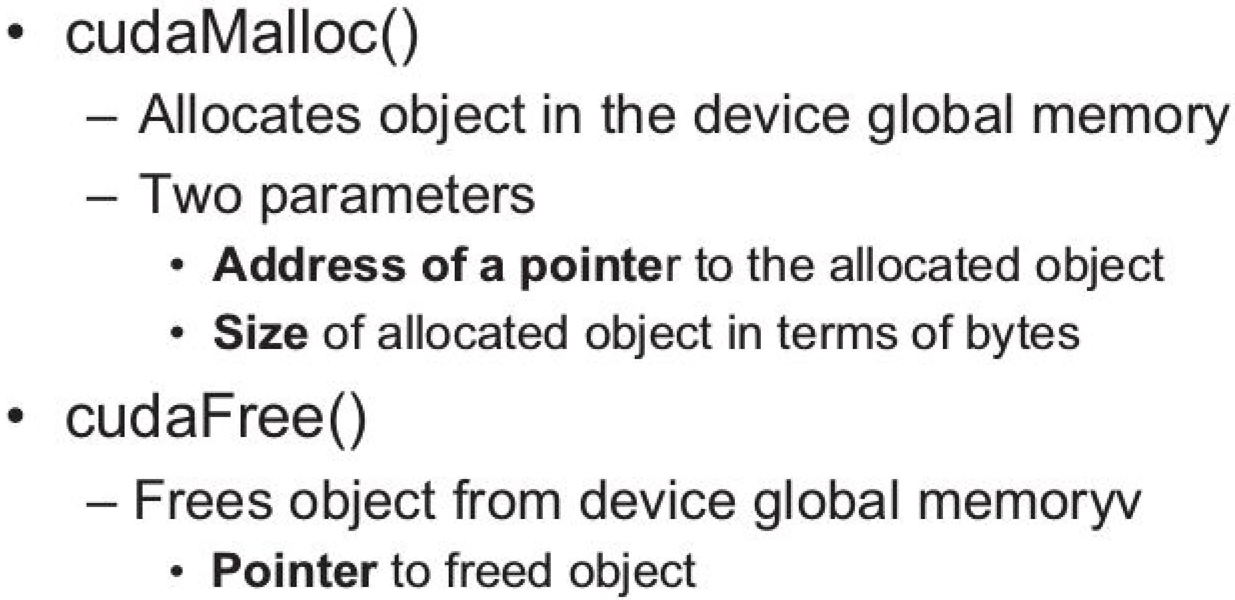
Code Scheme

Device Global Memory And Data Transfer



Memories block diagram

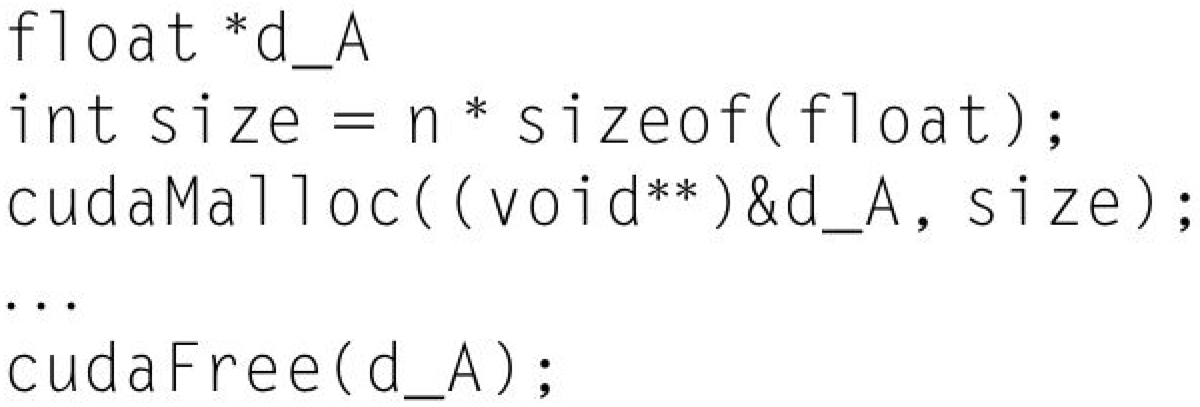
Device Global Memory And Data Transfer



- cudaMalloc()
  - Allocates object in the device global memory
  - Two parameters
    - **Address of a pointer** to the allocated object
    - **Size** of allocated object in terms of bytes
- cudaFree()
  - Frees object from device global memoryv
    - **Pointer** to freed object

CUDA API Functions to manage device memory

Device Global Memory And Data Transfer



```
float *d_A
int size = n * sizeof(float);
cudaMalloc((void**)&d_A, size);
...
cudaFree(d_A);
```

CUDA Malloc Example

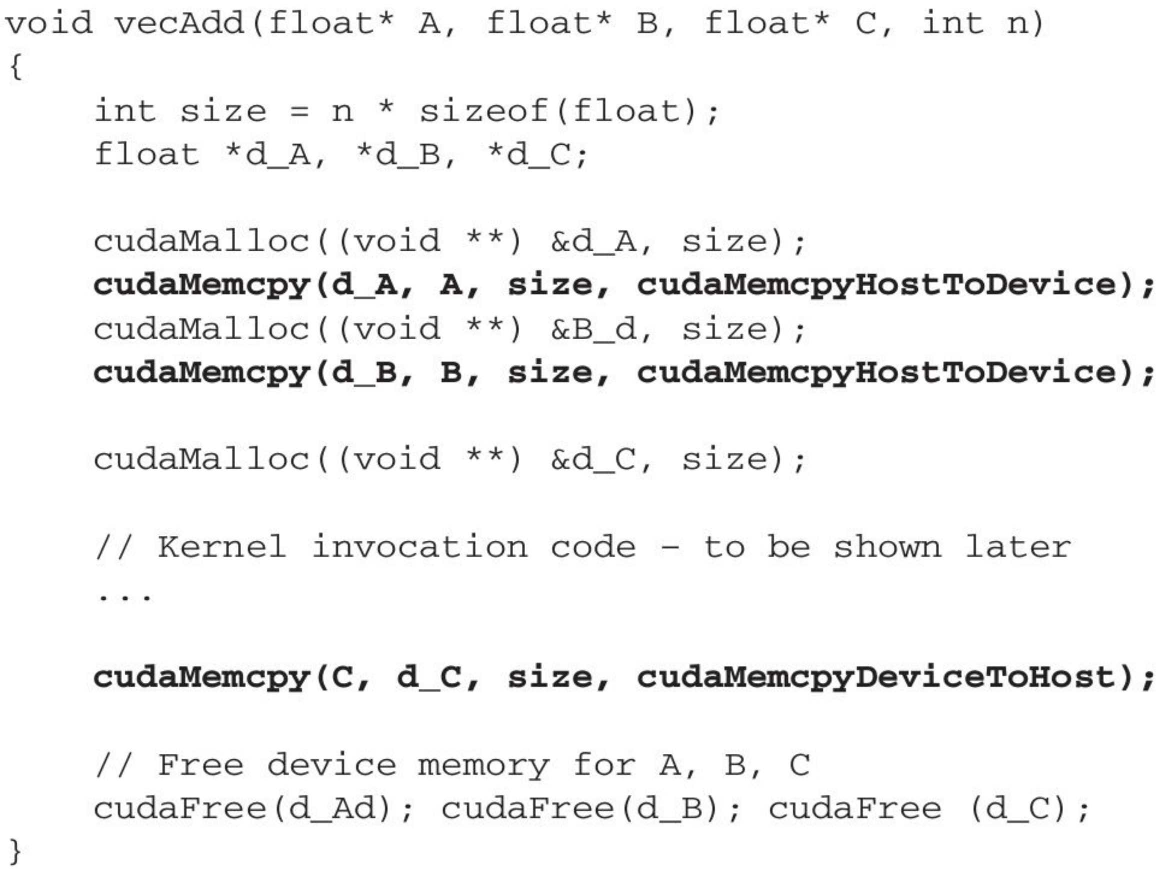
Device Global Memory And Data Transfer

cudaMemcpy()

- memory data transfer
- Requires four parameters
  - Pointer to destination
  - Pointer to source
  - Number of bytes copied
  - Type/Direction of transfer

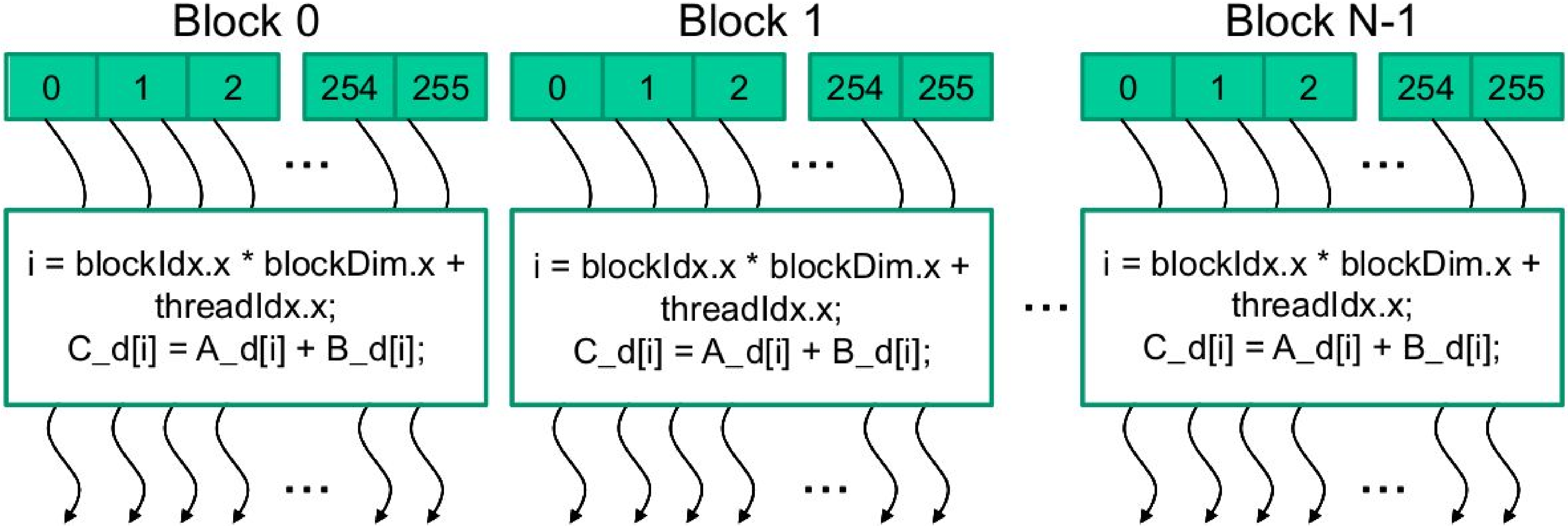
Device Global Memory And Data Transfer

A more complete version



```
void vecAdd(float* A, float* B, float* C, int n)
{
    int size = n * sizeof(float);
    float *d_A, *d_B, *d_C;

    cudaMalloc((void **) &d_A, size);
    cudaMemcpy(d_A, A, size, cudaMemcpyHostToDevice);
    cudaMalloc((void **) &B_d, size);
    cudaMemcpy(d_B, B, size, cudaMemcpyHostToDevice);

    cudaMalloc((void **) &d_C, size);

    // Kernel invocation code – to be shown later
    ...

    cudaMemcpy(C, d_C, size, cudaMemcpyDeviceToHost);

    // Free device memory for A, B, C
    cudaFree(d_Ad); cudaFree(d_B); cudaFree (d_C);
}
```

Kernel Functions and Threading



Block 0

0 1 2 254 255

…

```
i = blockIdx.x * blockDim.x +
threadIdx.x;
C_d[i] = A_d[i] + B_d[i];
```

…

Block 1

0 1 2 254 255

…

```
i = blockIdx.x * blockDim.x +
threadIdx.x;
C_d[i] = A_d[i] + B_d[i];
```

…

…

Block N-1

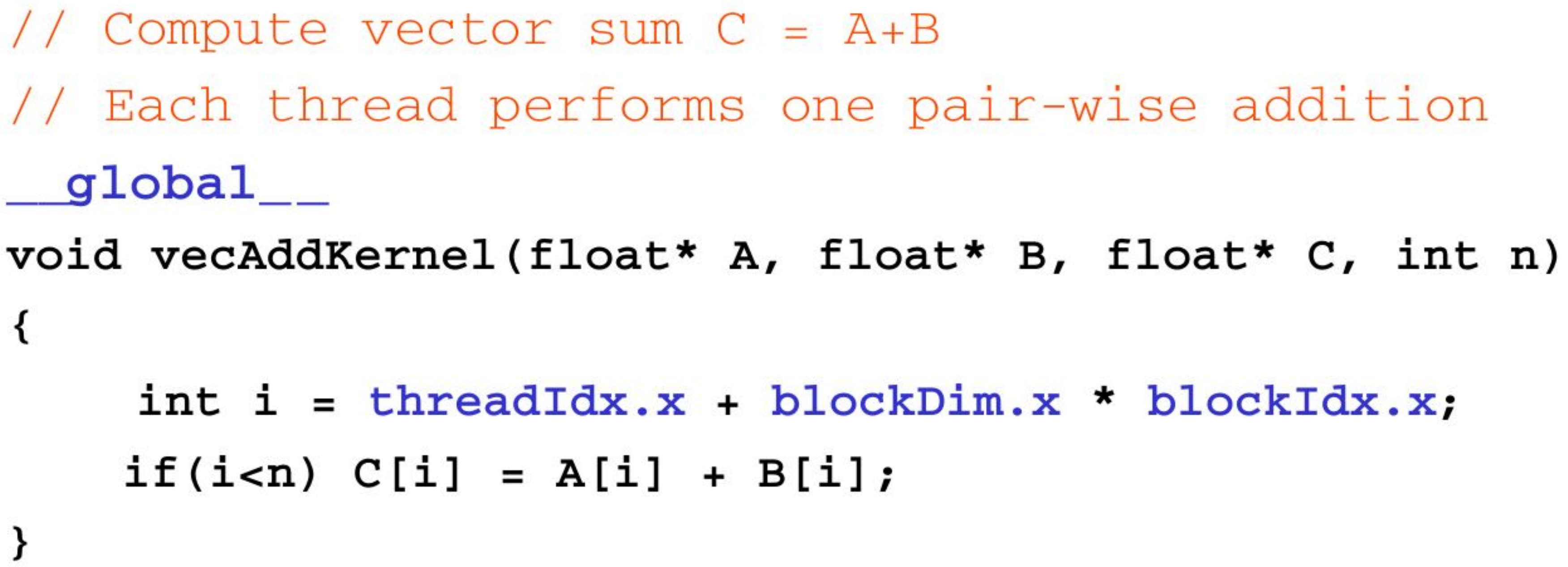
0 1 2 254 255

…

```
i = blockIdx.x * blockDim.x +
threadIdx.x;
C_d[i] = A_d[i] + B_d[i];
```

…

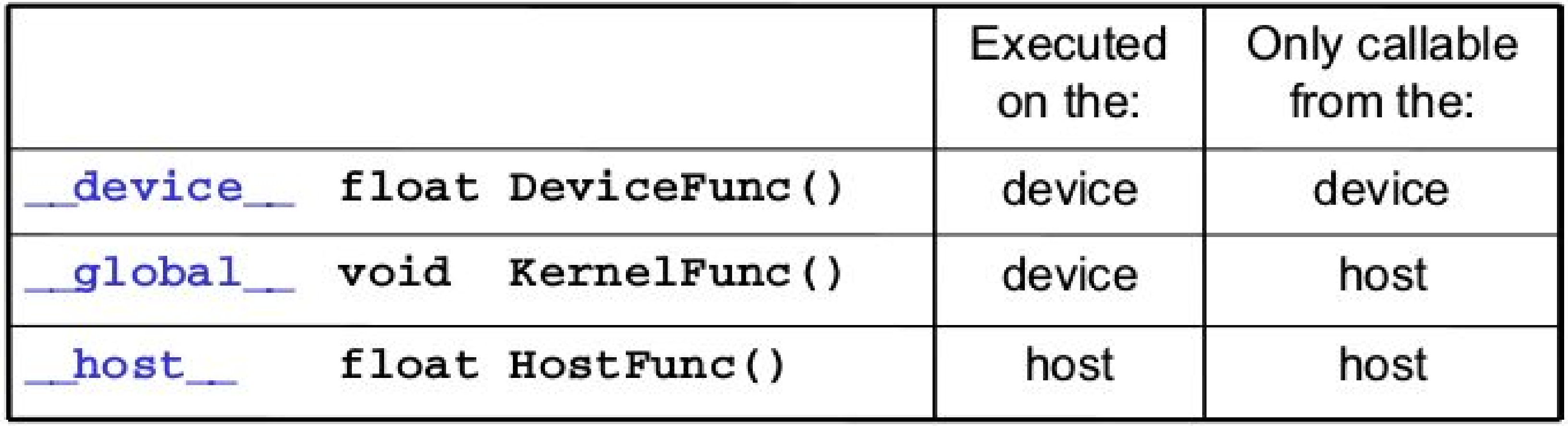
Kernel Functions and Threading



```
// Compute vector sum C = A+B
// Each thread performs one pair-wise addition
__global__
void vecAddKernel(float* A, float* B, float* C, int n)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
   if(i<n) C[i] = A[i] + B[i];
}
```

Vector Addition Kernel

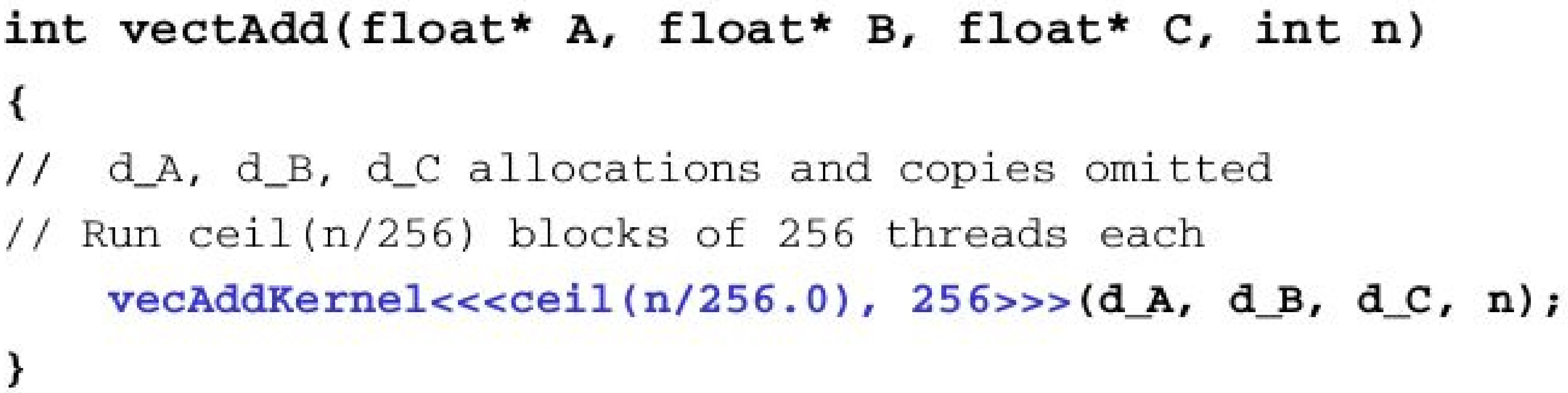
Kernel Functions and Threading



	Executed on the:	Only callable from the:
`__device__ float DeviceFunc()`	device	device
`__global__ void KernelFunc()`	device	host
`__host__ float HostFunc()`	host	host

CUDA C Keywords for Function Declaration

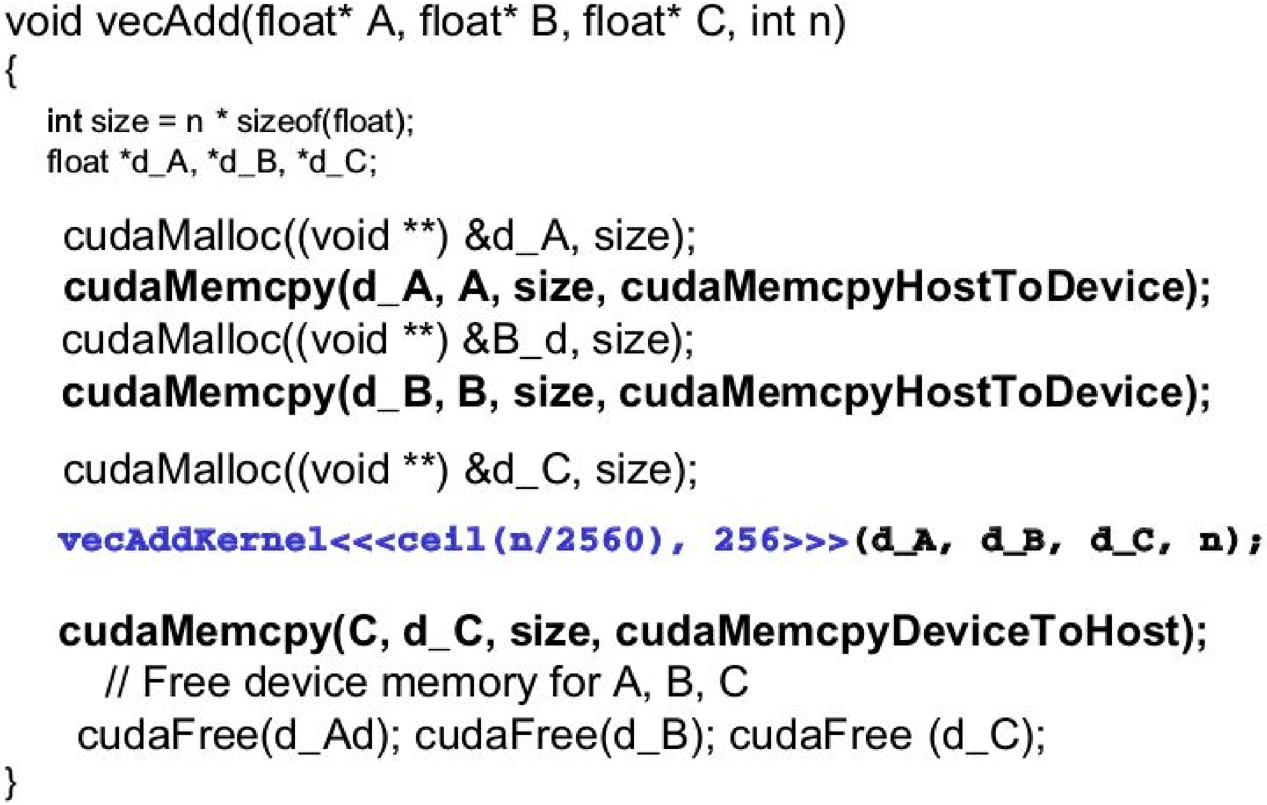
Kernel Functions and Threading



```
int vectAdd(float* A, float* B, float* C, int n)
{
//  d_A, d_B, d_C allocations and copies omitted
// Run ceil(n/256) blocks of 256 threads each
    vecAddKernel<<<ceil(n/256.0), 256>>>(d_A, d_B, d_C, n);
}
```

Kernel Launch Statement

Kernel Functions and Threading



```
void vecAdd(float* A, float* B, float* C, int n)
{
    int size = n * sizeof(float);
    float *d_A, *d_B, *d_C;

     cudaMalloc((void **) &d_A, size);
     cudaMemcpy(d_A, A, size, cudaMemcpyHostToDevice);
     cudaMalloc((void **) &B_d, size);
     cudaMemcpy(d_B, B, size, cudaMemcpyHostToDevice);

     cudaMalloc((void **) &d_C, size);

     vecAddKernel<<<ceil(n/2560), 256>>>(d_A, d_B, d_C, n);

     cudaMemcpy(C, d_C, size, cudaMemcpyDeviceToHost);
        // Free device memory for A, B, C
      cudaFree(d_Ad); cudaFree(d_B); cudaFree (d_C);
}
```

Complete VecAdd

NVIDIA
CUDA
adidas

THANKS

john@sirius.utp.edu.co